2

SIX-MONTH REPORT
OCTOBER 1, 1993 TO MARCH 31, 1994

INVESTIGATION OF MODULARLY CONFIGURED ATTACHED PROCESSORS WITH INTELLIGENT MEMORIES

GRANT NO. N00014-93-1-1343

This report has been divided into three parts:

1. Personnel (students employed to date).
2. Preparation (steps taken to prepare for completing objectives).
3. Progress (work completed toward accomplishing objectives).
4. Publications (publication of initial results).

1. Personnel

Master's student (began October 1, 1993)--studying MCM implementation of an MCAP.

Master's student (began December 1, 1994)--wrote the graphical interface for the simulator and is now studying algorithms.

Master's student (began January 16, 1994)--helped test and modify the program assembler needed by the simulator and is now studying algorithms.

Undergraduate student (began January 16, 1994)--testing the simulation portion of the simulator and writing the portion for outputting results.

Undergraduate student (began January 16, 1994)--testing the simulation portion of the simulator and embedding error checking into the simulator.

In addition, two Ph.D students have committed to working on wafer scale integration and design of the memory controllers and a post-doctorate has been hired at a 30% rate to help Dr. Sergio Cabrera on his study of algorithms.

2. Preparation

The preparation has consisted of attending the conferences:

Dr. Chang and a master's student: **IEEE International Conference on Wafer Scale Integration**, January, 1993, San Francisco, CA

A master's student (only the airfare was paid by contract), **SPIE Symposium on Electronic Imaging Science and Technology**, Feb., 1994, San Jose, CA

Dr. Singh and a master's student: **IEEE Multichip Module Conference**, March, 1994, Santa Cruz, CA

3. Progress

Objective 1 (register-level design of MCAP): The register-level design and preliminary estimates of power, number of connections, and so on is 80% complete (see attachments).

Objective 2 (architecture/algorithm case studies): The study of relative speeds and routing patterns as different algorithms and architectures are considered is just beginning.

Objective 3 (two memory controller designs): One of the designs is complete, but is awaiting the completion of the simulator for evaluation. The other design has not begun.

Objective 4 (technology evaluations): Preliminary work has concentrated on CMOS and MCM packaging. This phase is nearing completion and work involving high-speed technologies and wafer-scale integration is beginning.

Objective 5 (simulator development): Except for the memory controller components, the simulator has been completed and is currently being tested.

4. Publications

The attachments to this report have been submitted to the following conferences, respectively:

A. **Seventh International Conference on Parallel and Distributed Computing Systems**, October, 1994, Las Vegas, NV

B. **Sixth IEEE Symposium on Parallel and Distributed Processing**, October, 1994, Dallas,TX

Att: A

MCM IMPLEMENTATION OF MODULARLY CONFIGURABLE ATTACHED PROCESSORS[1]

Glenn Gibson, Vijay Singh, Sanjay Singh,
Yu-cheng Liu, Yi-Chieh Chang, and Sergio Cabrera
Electrical Engineering Department
The University of Texas at El Paso
El Paso, Texas 79968-0523

ABSTRACT

A new architecture for high-performance parallel attached processors is described in this paper. Based on this architecture, an attached processor can be implemented as multiple memory-to-memory pipelines, each being constructed with a class of fundamental components. The unique features are that the attached processor can be configured to match a set of algorithms and its memory controllers can be programmed to fit the access patterns required by the algorithms. As a result, high utilization of the processing logic for given sets of algorithms can be obtained. An example based on matrix multiplication is used for illustration. Finally, design issues related to the implementation of the attached processor based on an MCM technology are discussed.

Index Terms: Attached processor, interconnected pipeline, memory-to-memory pipeline, sustained execution rate, multichip module.

[1]The work reported in this paper was supported in part by the Office of Naval Research under Grant No. N00014-93-1-1343. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of the funding agency.

1 Introduction

An attached, or back-end, processor is a processing system that is connected to a host computer for the purpose of very quickly executing most of the overall system's computational tasks. In such an organization, "the host is a program manager which handles all I/O, code compiling, and operating system functions, while the back-end attached processor concentrates on arithmetic computation with data supplied by the host machine" [1].

Typical early attached processors were the AP-120B and FPS-164 made by Floating Point Systems, Inc., the IBM 3838, and the MATP made by Datawest, Inc. [1], [2], [3]. These attached processors all have their own data memories and transfer data between these memories and the main memories of their hosts using DMA data channels. They also include their own code memories where subprograms may be permanently stored or downloaded from their hosts. These subprograms are initiated by commands from the host and supervise the data flows from the attached processor's data memories, through the attached processor's processing elements, and back into the data memories.

Although the early attached processors included limited multiprocessing, the more recently implemented processing arrays are also controlled by a host (e.g., the PAX computer [4]) and are designed to perform most of the overall system's computational tasks. Therefore, these arrays and even the array processing portions of today's supercomputers, such as the Cray series [1], [3] could be interpreted as attached processors, although the host is then sometimes referred to as a front-end computer.

The specific purpose of an attached processor is to execute members of a set of algorithms very quickly. The broader the set of algorithms the more generally applicable the attached processor. The underlying goal of the designer is to efficiently utilize the hardware for as broad a set of algorithms as possible. However, for most current designs, the average sustainable execution rates have been found to be only 5% to 20% of their peak rates, which are determined by summing the maximum computational rates of the processing elements. For example, the sustainable rate for a Cray X-MP with four processors may be as low as 5% for some algorithms [5]. Also extensive evaluations of recent high-performance computations using Lapack are given in [6] and using NSA parallel benchmarks are given in [7]. Although some of the lost efficiency is necessitated by the algorithms, much of it is due to memory accessing and contention for shared resources in general, including internal buses.

Described in this paper is a class of high-performance attached processors called Modularly Configurable Attached Processors (MCAPs) which can attain quickness and high utilization through:

- Closely matching their architectures to the set of algorithms they are to execute.
- Overlapping of processing and memory accessing by using memory prefetching.
- Minimizing the movement of data.
- Using a high-speed technology with MCM or wafer scale implementations.

An MCAP is constructed from the component types specified in Sec. 2. These component types are such that each member of the class may include parallel processing, memory-to-memory pipelines, and be constructed in a building block fashion. They encompass routing components (including buses) as well as memory, control, and processing components. By overlapping processing with memory accessing and matching an architecture with a set of algorithms, it is predicted that the average sustainable rate for a specific set of algorithms can attain at least 60% of the peak rate. By defining components that are simple enough to be fabricated onto single low-density ICs, a high-speed technology may be used.

Much of an MCAP's efficiency is gained by distributing the instructions for the next algorithm (or algorithm phase) to the various components while the current algorithm (or phase) is executing. Once the algorithm begins, these instructions dictate the modes, routing patterns, prefetching patterns, and so on of the components receiving them. After an algorithm starts, each component operates more or less on its own except for responding to its handshaking signals. Efficiency is further enhanced by prefetching operands from the memory subsystems. Prefetching using programmed patterns avoids the misses that result from using ordinary caches.

Section 2 describes the architecture of the MCAP and the fundamental components required to construct an MCAP. Section 3 illustrates how to match an algorithm with a given MCAP architecture in order to attain a high sustainable rate of its peak performance. A major issue related to the implementation of MCAPs is the choice of semiconductor technology and packaging, which affect speed, gate density, power dissipation, and cost. The emphasis of implementation considerations given in this paper is on CMOS Multi-Chip Module (MCM) technology due to its ability to achive fast inter-chip communication. Section 4 discusses various design issues involved using the MCM approach to implement the MCAPs. Such considerations include transistor count, loading, estimate of speed, and power dissipation.

2 MCAP Architecture

An MCAP is an attached processor that is constructed entirely from a standard set of connections and components. This standard set consists of three types of asynchronous connections and twelve types of components. The definitions of the connection and component types provide a standard set of rules that allow the components to be easily configured in different ways to construct attached processors that can efficiently perform different sets of algorithms.

An MCAP has exactly one instruction component and it is connected to a memory component for storing instructions. Most of this memory component is a ROM that contains the subprograms needed to execute the algorithms, but some of it is a RAM that can receive instructions (those that initiate the subprograms) from the host.

An MCAP operates by drawing an instruction stream from the memory component into the instruction component. The instruction component uses internal instructions in the stream to form external instructions that are then distributed to the other non-memory components through the MCAP's (one and only) bus component. The instruction stream is illustrated in Fig. 1. Note that all components in the instruction stream include input instruction queues. When the non-memory components have received all of the instructions needed to perform an algorithm, they automatically prefetch the data from the memory components, route the data to and from the processor components and store the results back into the memory components. A typical data stream is depicted in Fig. 2. It is seen that all non- memory components have input data queues. DMA units built into some controller components, which are the components that supervise all memory accessing, are used to automatically transfer data between the host's main memory and the MCAP's memory components while the algorithm is executing. Also, the instruction and data streams are separate, thereby allowing the instructions needed for the next algorithm to be distributed while the current algorithm is executing.

The three types of connections are referred to as memory, instruction, and data connections. All connections are asynchronous and, therefore, must include handshaking lines as well as data and, perhaps, address lines. Each memory component is connected to its controller component by a single memory connection that consists of a bidirectional data bus, a unidirectional address bus and a Request (Req)/ Acknowledge (Ack)/Memory Request (MReq) handshaking triplet.

Instruction connections are for passing instructions from the instruction component to the bus component and from the bus component to one of the other non-memory components. An instruction connection consists of unidirectional instruction and address buses and a Req/Ack handshaking pair. The component that is to receive the instruction is indicated by the a component number on the address bus. A transfer is initiated when the sending component puts an address on the address bus, an instruction on the instruction bus and begins the handshaking. Except for the connections to memory components, all connections used to transfer data are data connections. They are used to pass data to and from the processors and consist of only a unidirectional data bus and a Req/Ack pair. A data transfer consists of placing data on the data bus and initiating the handshaking. Except for a write to a memory component, all transfers include the latching of an instruction or datum into a queue at the receiving end.

The twelve types of components are divided into six categories as indicated below:

Instruction
Bus
Memory
Processor
    Elementary–one input, one output

Two-input–two inputs, one output
Comparator–two inputs, one output plus special outputs

Router

Join–multiple inputs, one output
Fork–one input, multiple outputs
Link–multiple inputs, multiple outputs

Controller

RAM–internal to MCAP, no partitions
Single-access–internal to MCAP, has partitions
Dual-access–connects to main memory, has partitions

As mentioned earlier, an MCAP contains one memory component for storing instructions, one instruction component for executing internal instructions and forming external instructions, and one bus component for distributing the instructions. An MCAP may contain several controller, router, and processor components and several other memory components for storing data. However, the other memory components can be connected to controller components only. Only controller components are capable of being programmed to prefetch data from and deposit data into data memory components. Although the instruction memory component or a dual-access component can be connected to the host system, all other components can be connected to the MCAP's components only.

Each non-memory component that is used during the execution of an algorithm contains an instruction input queue, one or more data input queues, and control logic that includes a number of registers. The instructions for an algorithm received by a component fill these registers and then the register contents dictate the activity within the component while the algorithm is executed. They determine the component's mode and, for a routing component, the patterns for accepting inputs and distributing outputs. For a controller component, they determine the memory partitions, DMA accessing patterns, and patterns for prefetching the operands needed by the algorithm.

Each of the components that receives instructions contains a Number of Operands Output (NumOpsOut) register that is always the last register filled before the component begins its part in the execution of the algorithm. Each time the component outputs an operand, the NumOpsOut register is decremented. When the NumOpsOut register becomes zero, the component has completed its part in executing the current algorithm. It may then distribute new values, those needed for the next algorithm, from its instruction input queue to its registers. This cycle may continue indefinitely. Except for reacting to the handshaking (i.e., Req and Ack) signals in its connections, each component acts independently. The data is input to a data queue through an input connection, processed or routed through a bus, and output through an output connection. Because separate queues are used to input instructions and data, the instruction and data streams are completely separate.

The processor components are used for performing unary and binary arithmetic/logic operations. There are three types of processor components. There are one-input elementary (E) components, two-input (T) components, and comparator (C) components. These components contain only two registers, a mode register and a NumOpsOut register. The mode register dictates the actions taken by the component and the NumOpsOut register gives the total number of operands that is to be output before the current algorithm is completed. Both the E and T components may be used for either unary or binary operations, depending on the mode. When an E component is used for a binary operation it must, of course, input both operands through its single input connection. A T component performing a unary operation would use only one of its two input connections.

A C component is similar to a T component, but has two special sets of lines connecting it to the instruction component. There can be only one C component in an MCAP. As usual, its current function is determined by its mode. One of its functions is to simply compare two inputs and set relational flags that are then transmitted to the instruction component over one set of the special lines. When performing comparisons, there are no outputs other than the flag outputs. The C component can, however, also determine the maximum or minimum of a sequence of numbers. In this case, the second set of special lines is used to output the index of the maximum or minimum to the instruction component. The maximum or minimum is output on the output data connection.

Routing components are for directing data along the proper paths. There are three types of routing components, join (J) components with more than one input and one output, fork

(F) components with one input and more than one output, and link (L) components with more than one input and more than one output. In addition to the mode and NumOpsOut registers, they contain registers for dictating their input and output patterns while the current algorithm is being executed. F and L components may include broadcasting in their output patterns. J and F components may be used in conjunction with T and E components to form pipelines with feedback that can accumulate sums.

There are three types of controller components, RAM (R) components, single-access (S) components, and dual-access (D) components. All controller components are for automatically retrieving operands from and storing results in their associated memory components. In addition, a D component contains DMA units for communicating with the host's main memory. All controller components have an output data connection for outputting operands to the remainder of the MCAP and an input data connection for inputting results from the MCAP. Therefore, they must be capable of handling both an output data stream and an input data stream. A queue is inserted in each of these data streams. A D controller also has memory connections between its DMA units and the host's main memory.

A significant difference between the controller components and the other programmable components is that a Number of Operands In (NumOpsIn) register as well as a NumOpsOut register must be included. The NumOpsIn register serves the same purpose for the input data stream as NumOpsOut does for the output stream. An S component differs from an R component in that its memory may be divided into partitions that consist of blocks of memory having consecutive addresses. The memory components are interleaved so that the partitions, because they occupy consecutive addresses, are spread across the components. In addition to the mode, NumOpsOut and NumOpsIn registers, an S component contains registers for specifying the patterns for accessing the partitions and a set of registers for each partition for specifying the pattern of accesses within the partition.

That portion of a D component that communicates with the MCAP is similar to an S component except for the inclusion of a window in each partition. A window is a set of memory locations with consecutive addresses whose base address increments after each repetition of a pattern. The purpose of the window is to separate the communication with the MCAP from the communication with the host's main memory. Data that is output to or input from the MCAP must involve accesses that are within the window and main memory transfers must involve accesses that are outside the window. Because a partition is treated as a circular memory, the location with the highest address in the partition is considered to be adjacent to the one with the lowest address and the window is considered to move in a circle.

An example architecture is given in Fig. 3. Its processing subsection includes a comparator (C component), a negator (E component), a reciprocator (E component), a set of four pipelined adders capable of accumulation, and a set of four pipelined multipliers. Each adder or multiplier is constructed of four stages (a T component followed by three E components). All communications to and from the processing components are through six L components, three on each side of the processor. J and F components are provided to allow flexible use of the L components. Also, to allow for accumulation there is a feedback connection between the F component at the output from each adder and the J component at the input to the adder. There is a D component to provide intermediate memory and a connection to main memory. The S component provides internal storage.

3 Matching Algorithms to Architectures

In order to efficiently use the available logic and interconnections, an architecture must be carefully matched to an algorithm or set of algorithms. This involves a study relating the flows, storage and processing of the data required by the algorithm(s). Clearly, there is no point in increasing the speed of a processing subsystem if the current interconnections and memory hierarchy are inadequate to support the processing (or vice versa). But a good balance for one algorithm may not be a good balance for a different algorithm. What is needed is a satisfactory tradeoff for the work mix expected of a system and a means of evaluating the design parameters chosen.

Space allows only a single example, so let us consider the computation that most frequently occurs in computationally intense algorithms, matrix multiplication. Let us examine how the MCAP in Fig. 3 could be analyzed relative to the algorithm $AB = C$ using the middle product method [3] where A, B and C are $n \times n$ matrices. Fig. 4 shows the required flow of data through the MCAP. The variable m is the number of rows that can be simultaneously stored in each of the D and S component memories. The expressions give the total numbers of operands transferred between the major subsystems.

The algorithm consists of the computations

$$\sum_{i=1}^{n} a_{ij}\, B_j = C_j \qquad i = 1, \ldots, n$$

where the a_{ij}s are the elements of A, the B_js are the rows of B, and the C_is are the rows of C. The algorithm proceeds by storing the first m elements of the first column of A and the first m rows of B in the D component's memory. Then the products $a_{i1}B_i$, for $i = 1, \ldots, m$, are formed and stored in the S component. Next, the first m elements of the second column of A are brought into the D component and the products $a_{i2}B_i$ are formed and added to the corresponding previous products, with the results being returned to the S component. This is repeated $n/m - 1$ times, but the last time the product totals, which are the first m rows of C, are put in the D component and then output to main memory. The entire process is repeated n/m times. Overlapping can be used to reduce the required time.

By matching this algorithm with the architecture in Fig. 3, it is seen that each adder and multiplier must perform approximately $n^3/2$ operations and each link on the left and two of the links on the right must perform approximately n^3 transfers. (The third link on the right is not be needed.) The approximate numbers of accesses to the S component, D component and main memory are about $2n^3, n^3(1 + 1/m)$ and n^3/m, respectively. If T is the per stage processing time of the multipliers, then T should also be the per stage processing time of the adders and $T/4$ should be the transfer time of the links. The access times of the S component, D component and main memory should be $T/8, mT/4(m+1)$ and $mT/4$, respectively for both reads and writes. For T = 40 ns and m = 8, the link transfer time should be 10ns and the average memory access times should be 5 ns, 9 ns and 80 ns. The computation rate would be 200 Mflops per second. If the MCAP were put into an MCM or wafer and memory interleaving were used, these times would certainly be within the capability of current HCMOS technology. (The join and fork components were ignored in this discussion because the communication times are dictated by the slower link components.) BiCMOS and GaAs could produce proportionately faster processing, memory and memory controller components, but, as seen in the next section, increasing the speed of the link components is a more challenging problem.

Except for the unused link component, the design would utilize the link and processor components over 95% of the time while performing a matrix multiplication. In contrast, note that matrix addition would utilize these components only about 50% of the time on the average with the S, multiplier and some of the routing components not being used at all. This contrast points out the need for different designs for different algorithms and the need for compromise when a set of algorithms must be executed on the same architecture.

4 MCM IMPLEMENTATION CONSIDERATIONS

Since the signal delays associated with a PCB implementation are expected to be prohibitively excessive, it is thought that the fabrication of an MCAP in a Multi-Chip Module (MCM) configuration or Wafer Scale Integration (WSI) are the only realistic alternatives for attaining high-performance. Some important design considerations for implementing an example MCAP architecture in MCM configuration are presented in this section.

Fig. 5 show's a layout for an MCM implementation of the example architecture. In designing this layout, we aimed toward minimizing chip to chip interconnections, maximizing interconnection densities, and using a parallel architecture. Other factors of importance are ground and power plane generation and physical design verification. The amount of heat generated is directly dependent on the type of substrate (MCM's are classified according to the substrate technology; MCM-C, MCM-D, and MCM-L), selection of bonding and placement of chips. Parasitics on the interconnects, inductances on the power lines and the I/O pin limitation are other important considerations.

4.1 The transistor count

In estimating the total number of transistors required to build the proposed MCAP, we made the assumption that the technology used is high-speed CMOS. CMOS was picked as the first benchmark technology because of its commercial maturity. In future, we paln to evaluate other faster technologies like GaAs in comparison with the COMS benchmark. As an example, let us consider a pipelined 64-bit floating point adder with four stages. It has:

1. Nine 64-bit registers with 4032 transistors (7 transistors per bit for a dynamic latch).
2. Seventy-four 2-input XOR gates with 592 transistors.
3. One hundred and twenty-six 2 to 1 MUX's with 504 transistors.
4. Two 11-bit adders with 528 transistors.
5. One 52-bit adder with 1248 transistors.
6. A 64-bit leading zero detector with 5000 transistors.
7. Two 52-bit barrel shifters with 4000 transistors.
8. Rounding and other control logic taking 6500 transistors.

The total is 23K transistors for an adder. By having four pipelined stages, we can achieve stage delays of less than 20 ns [8]. This delay is of course expected to be even smaller for faster technologies like GaAs. Similarly, we can evaluate the number of transistors for a pipelined 64-bit floating point multiplier (using an optimized, modified Booth's algorithm) and arrive at a total of 58K transistors. Again with four pipelined stages, the delay per stage is less than 20 ns [8]. Following this procedure, the transistor count for the rest of the elements in the MCAP are calculated and Table 1 gives the count for the various components. A figure of approximately ten million is reckoned as the transistor count to build the whole MCAP.

In the proposed architecture, the bottle neck is the communication through the LINK elements because of their high fan-out and relatively large interconnection distances. This means that the output buffers for these elements must be relatively large. Next, we present the delay, power and area calculations for the output buffers as functions of *fan-out* (F) and *interconnection length* (ℓ).

1. The input capacitance of a gate including the lead and ESD capacitance is $C_{in} = 1$ pF.
2. The width of the metal conductor used for an interconnection is $w = 25\mu$m.
3. The capacitance of the metal conductor is $C_{m1} = 30$ aF/μm^2.
4. The sheet resistance of the metal is $R_s = 0.05\Omega/\square$.
5. The feature size is $\lambda = 0.5\mu$m.

4.2 Load capacitance

For the load capacitance

$$C_l = C_{int} + F \times C_{in}, \tag{4.1}$$

with

$$C_{int} = w \times \ell \times C_{m1} = (0.025) \times \ell \times 30 \times 10^{-18} \times 10^6 \text{ pF} = 0.75 \times \ell \text{ pF}$$

where ℓ is in mm and $C_{in} = 1$ pF. Therefore,

$$C_l = (0.75 \times \ell + F) \text{ pF} \tag{4.2}$$

The resistance of the interconnect is

$$\begin{aligned} R_{int} &= R_s \times (\ell/w) = 0.05 \times (\ell/0.025) \\ &= 2 \times \ell \ \Omega \end{aligned} \tag{4.3}$$

Thus, a LINK element with a fan-out of 19 and with an average interconnection length of 2 cm has load capacitance of 34 pF.

4.3 Average delay

It is known that, in general, the minimum size of a logic gate has a W/L ratio of 2. So, we start with a ratio of 2 and go to higher values in stages in order to drive a load within a short time. By dividing the buffering stages into the number of buffers with increasing W/L, optimum speeds can be achieved. It has been found that a stage ratio of 3 [9] gives best results. Also, the optimum number of stages is

$$N = 0.91(\ln C_l + 4.19) \tag{4.4}$$

where N is truncated to the nearest integer.

Using the optimum number of stages, the average delay is

$$T_{avg} = 0.484(N-1) + 5C_l/3(N-1) + 0.076 \text{ ns} \tag{4.5}$$

The plot of T_{avg} as a function of F and ℓ is shown in Fig. 6. For the example with F = 19 and ℓ = 20 mm, the delay time is seen to be 3.2 ns.

4.4 Buffer area

A simple inverter with $(W/L)_n = (W/L)_p = 2$ will need an area of 66 μm^2. A buffer with equal rise (t_r) and fall (t_f) times requires $(W/L)_p = 2(W/L)_n = 4$ and the area is going to be 150 μm^2. The total area of the buffer depends on the number of stages and, hence, is a function of F and ℓ. We have

$$\text{Area} = 66 + 3[14(N-1) + 36(1 + 3 + 3^2 + \cdots + 3^{N-2})] \approx 55 \times 3^{N-1} \ \mu m^2$$

The area as a function of F and ℓ is plotted in Fig. 7. For $F = 19$ and $\ell = 20$ mm, the area is $40 \times 10^3 \mu m^2$.

4.5 Power dissipation in the buffer

In CMOS, most of the power is dissipated during switching and, hence, dynamic power is approximately equal to the total power. The dynamic power is

$$P_d = C_T \times v^2 \times f_{avg} = 25(C_l + C_{buff})/T_{avg}$$

where $C_{buff} = 0.0152\,(3^{N-1})$ pF

Since the design of an MCAP uses asynchronous communication, the transfers over a LINK component involves the return of an acknowledge signal and the transmission of an output enable signal. It is estimated that the transfer rate may be as high as $f = 1/2T_{avg}$ Hz. For $F = 19$ and $\ell = 20$ mm, the total power dissipated by buffer is 175 mW (see Fig. 8).

4.6 Thermal management

There have been successive revolutions in device technologies, proceeding from TTL, ECL and NMOS to the recent high-speed CMOS, BiCMOS and GaAs. Three to five orders of magnitude reduction in minimal feature size, an order of magnitude in the characteristic chip dimension and, more importantly, a significant drop in the transistor switching energy from more than $10^{-9}\,J$ to nearly $10^{-16}\,J$ [10]. Power dissipation, in a leading edge bipolar chip, with 1 cm^2 area has reached 20 - 25 W, and based on a short term extrapolation of current trends in the packaging technology, it may well be anticipated that the power dissipation might approach more than 100 watts for 50 million transistors on the same 1 cm^2 area with a switching speed of 10 ps [10]. After comparing various existing VLSI modules in terms of thermal parameters [10], the value of heat flux, Q = 25 W/cm^2 seems to be reasonable for air cooling. Considering again the critical LINK element in the MCAP, we estimate Q = 14.24 W/cm^2 to drive 2 cm of interconnect and 19 gates. It is reasonable to expect, therefore, that for a MCAP architecture implemented in MCM, air cooling would be sufficient.

5 Conclusions

The architecture to implement a class of high-performance attached processors, which can be modularly configured to match given sets of algorithms, has been presented. The high utilization rate of the processing components is achieved mainly by (1) minimizing the movement of intermediate results; (2) prefetching almost all operands using intelligent memory controllers; and (3) reconfiguring (through programming) the interconnection of the processing components to match the needs of a given algorithm.

An example MCAP architecture was evaluated for MCM implementation. Because of its commercial maturity, the CMOS technology was picked as the first (benchmark) technology to be evaluated. Transistor count for implementing the MCAP was estimated at 9.85 million. In the proposed architecture, the bottleneck is the communication through the LINK elements because of their high fan-out and relatively large interconnection distances. For the LINK element output buffers, delay, power and area calculations were made as functions of fan-out and interconnection length. For example, a LINK element with a fan-out of 19 and an average interconnection length of 2 cm has a load capacitance of 34 pF, has a delay time of 3.2 ns, occupies an area of 40,000 μm^2 and dissipates 175 mW of power. Heat flux was estimated at 14.2 W/cm^2, which leads us to believe that air cooling will be sufficient for this MCAP architecture implemented in MCM.

Further improvements in MCAP performance could be obtained by: (1) Reducing the minimal feature size to 0.5 μm or 0.2 μm, (2) Minimizing the chip to chip spacing by mounting the chips on two sides, (3) Employing a higher speed technology like GaAs (HEMT's), (4). Perhaps, using Wafer Scale Integration.

ACKNOWLEDGEMENTS

We would like to acknowledge the contributions of Stephen Senyszyn and Dan Doran to design of the MCAP, particularly those related to the instruction synchronization mechanism.

References

[1] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, New York, 1985.

[2] J. A. Swanson, G. R. Cameron, and J. C. Haberland, "Adapting the ansys finite-element analysis program to an attached processor," *IEEE Computer*, vol. 16, no. 6, pp. 85–91, 1983.

[3] R. Hockney and C. Jesshope, *Parallel Computers 2*, Adam Hilger: Bristol, England, 1988.

[4] T. Hoshino, *PAX Computer: High-Speed Parallel Processing and Scientific Computing*, Addison-Wesley Publishing Company: Reading, Massachusetts, 1985.

[5] J. H. Tang and E. S. Davidson, "An evaluation of Cray I and Cray X-MP performance on vectorizable Livermore FORTRAN kernels," *Proc. 1984 Int'l Conf. on Supercomputing*, pp. 510–518, 1988.

[6] J. Dongarra, "Linear algebra libraries for high-performance computers: A personal perspective," *IEEE Parallel & Distributed Technology*, pp. 17–24, February 1993.

[7] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon, "Nas parallel benchmark results," *IEEE Parallel & Distributed Technology*, pp. 43–51, February 1993.

[8] H. Nakano and et. al., "An 80 MFLOPS (peak) 64-bit microprocessor for parallel computer," *IEEE J. Solid-State Circuits*, vol. SC-27, no. 3, pp. 365-372, 1992.

[9] K. E. N. Weste, *Principles of CMOS VLSI design - A systems perspective*, Addison Wesley, 1993.

[10] A. B. Cohen, "Thermal management of air and liquid cooled MCM's," *IEEE Trans on Components, Hybrids, Manufacturing Technology*, vol. CHMT-10, no. 2, pp. 159–175, 1987.

ELEMENT	DESCRIPTION	# OF TRANSISTORS
MEMORY ELEMENT	HAS 4K EACH OF RAM AND ROM	1.84M
INSTRUCTION	HAS 8 WORDS OF FIFO	10.0K
BUS	HAS 8 WORDS OF FIFO	10.0K
ELEMENTARY	HAS 8 WORDS OF FIFO	10.0K
TWO INPUT	HAS 8 WORDS OF FIFO	12.0K
JOIN	3 INPUTS AND FIFO OF 8 WORDS	14.0K
FORK	3 OUTPUTS AND FIFO OF 8 WORDS	07.0K
LINK	4 INPUTS AND 19 OUTPUTS	25.0K
STATIC RAM	16 ELEMENTS OF 1K EACH	6.30M
SINGLE ACCESS CONTROLLER	CONTROLS 8 MEMORY ELEMENTS	11.0K
DUAL ACCESS CONTROLLER	CONTROLS 8 MEMORY ELEMENTS AND 3 DMA CHANNELS	41.0K
COMPARE	SENDS OUT FLAGS AND INDICES	25.0K
RECIPROCATE	USING CONVERGENCE METHOD	50.0K
NEGATE	INVERT THE SIGN BIT	01.0K
F.P ADDER	USING CLA'S, BARREL SHIFTERS........	23.0K
F.P MULTIPLIER	USING MODIFIED BOOTH'S ALGORITHM	61.0K
MCAP	**Total number of Transistors**	**9.85 Million**
MCM	**With 50 Chips and 300 I/O's**	**9.85 Million**

Table 1. Transistor count for the various MCAP components

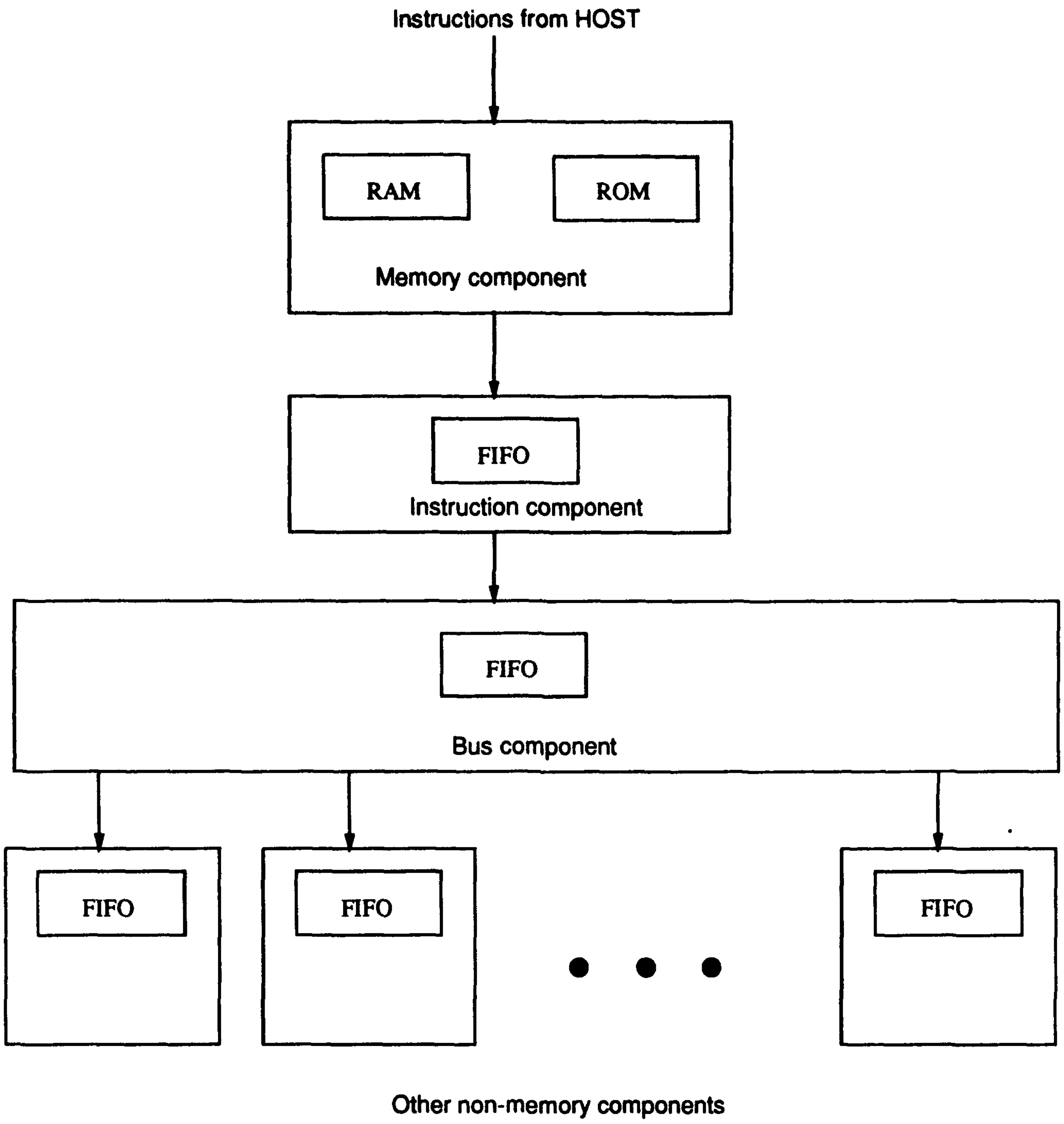


Fig. 1. The instruction stream

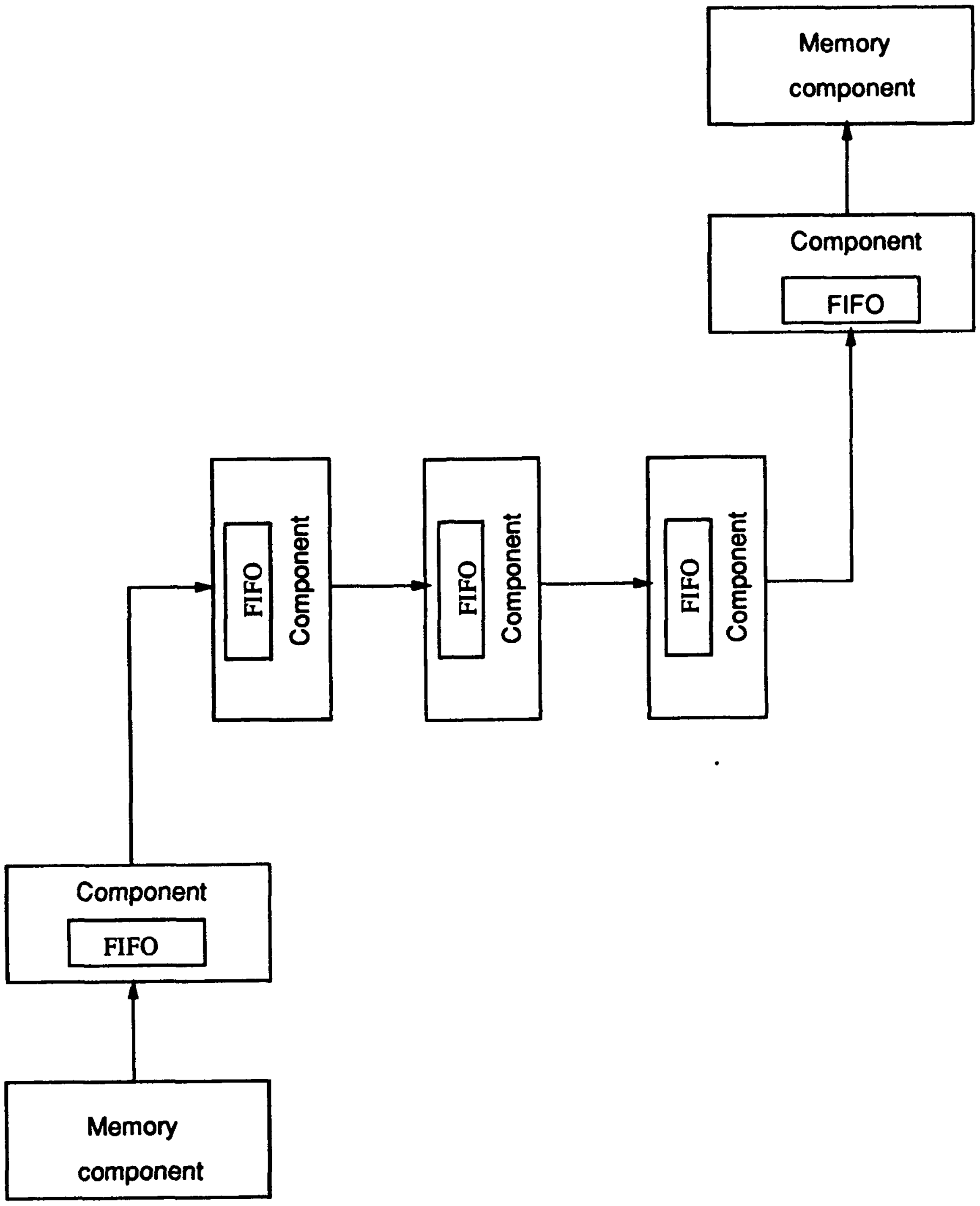


Fig. 2. Typical data stream

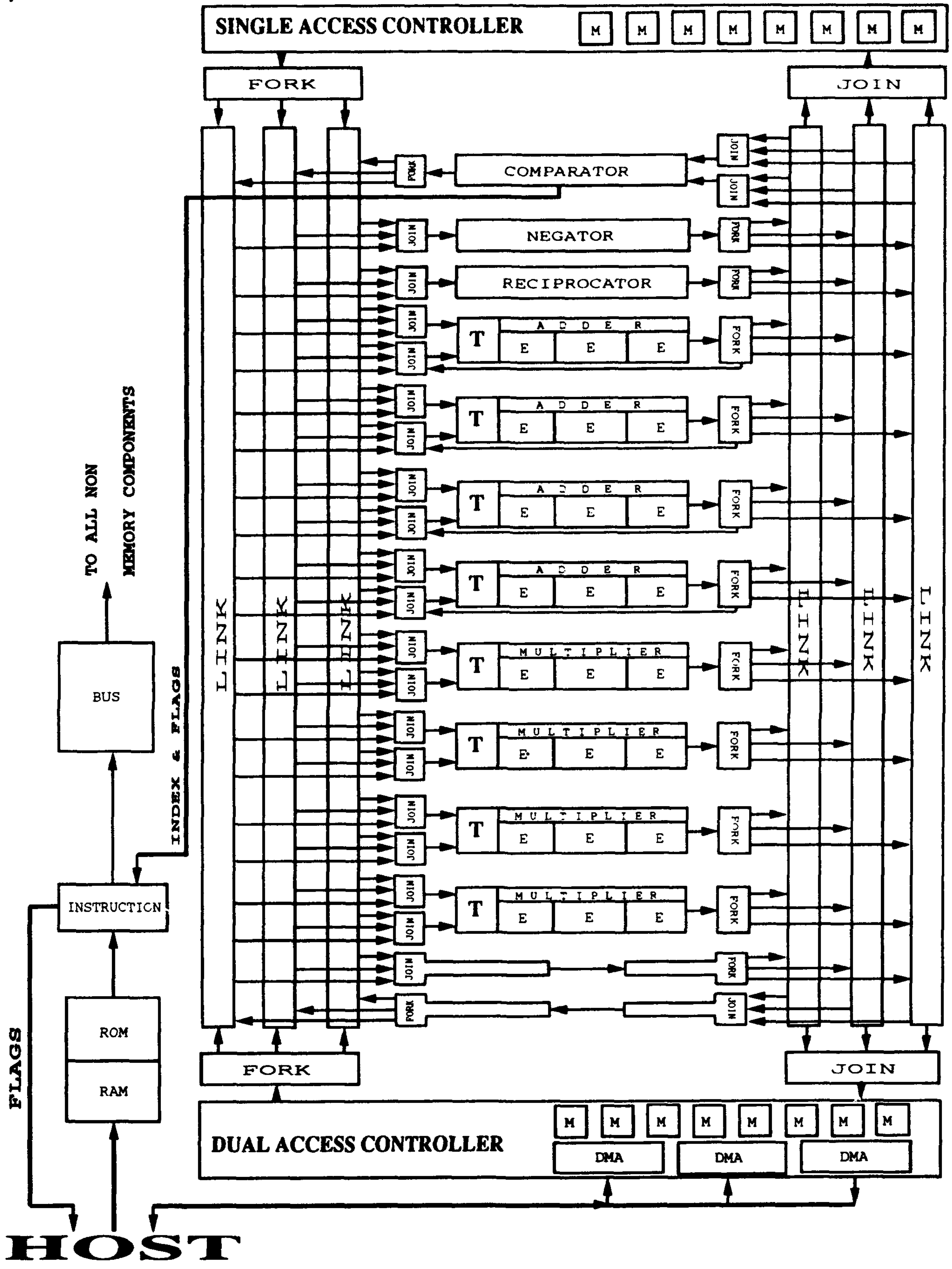


Fig. 3. An Example MCAP Architecture

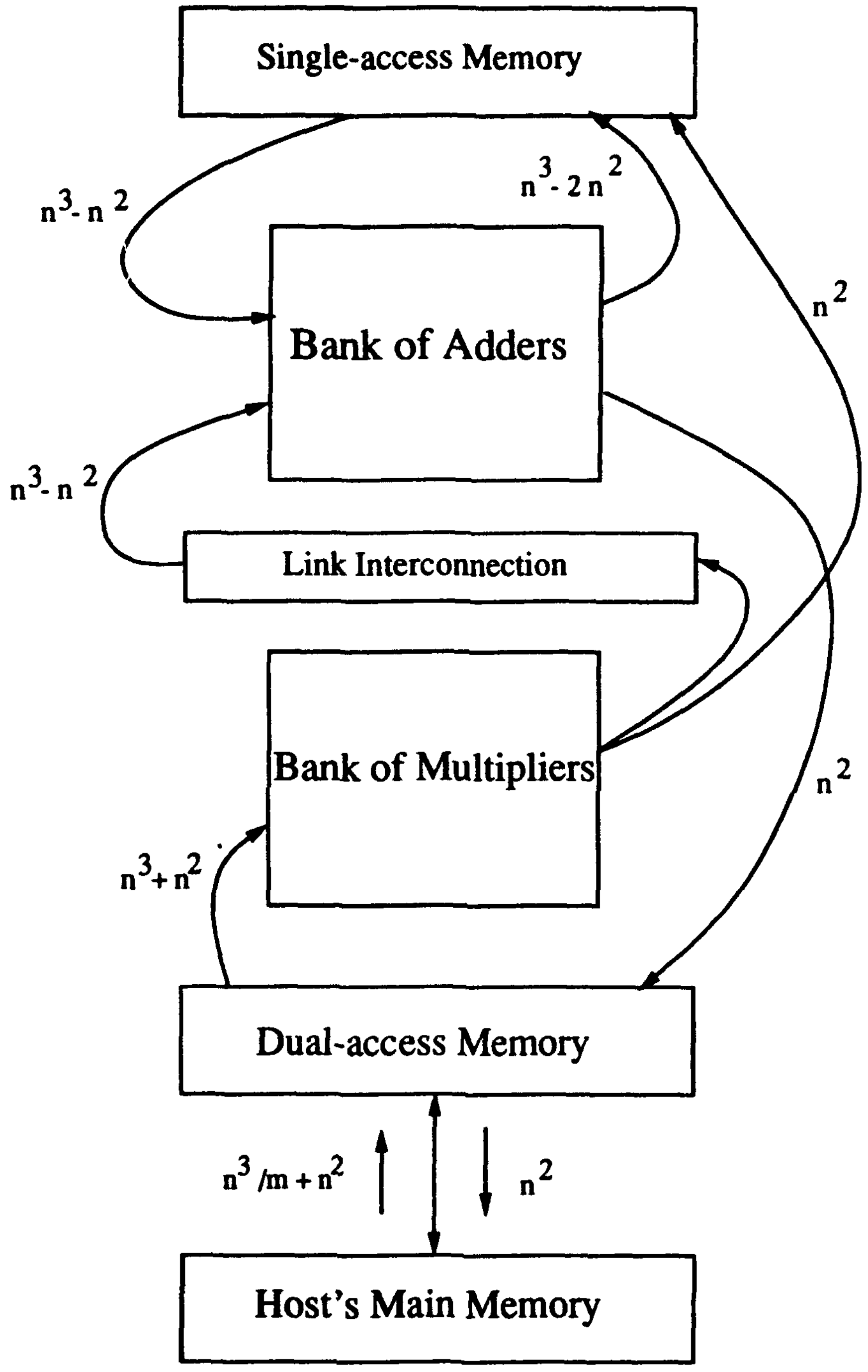


Fig. 4. Data flow for matrix multiplication

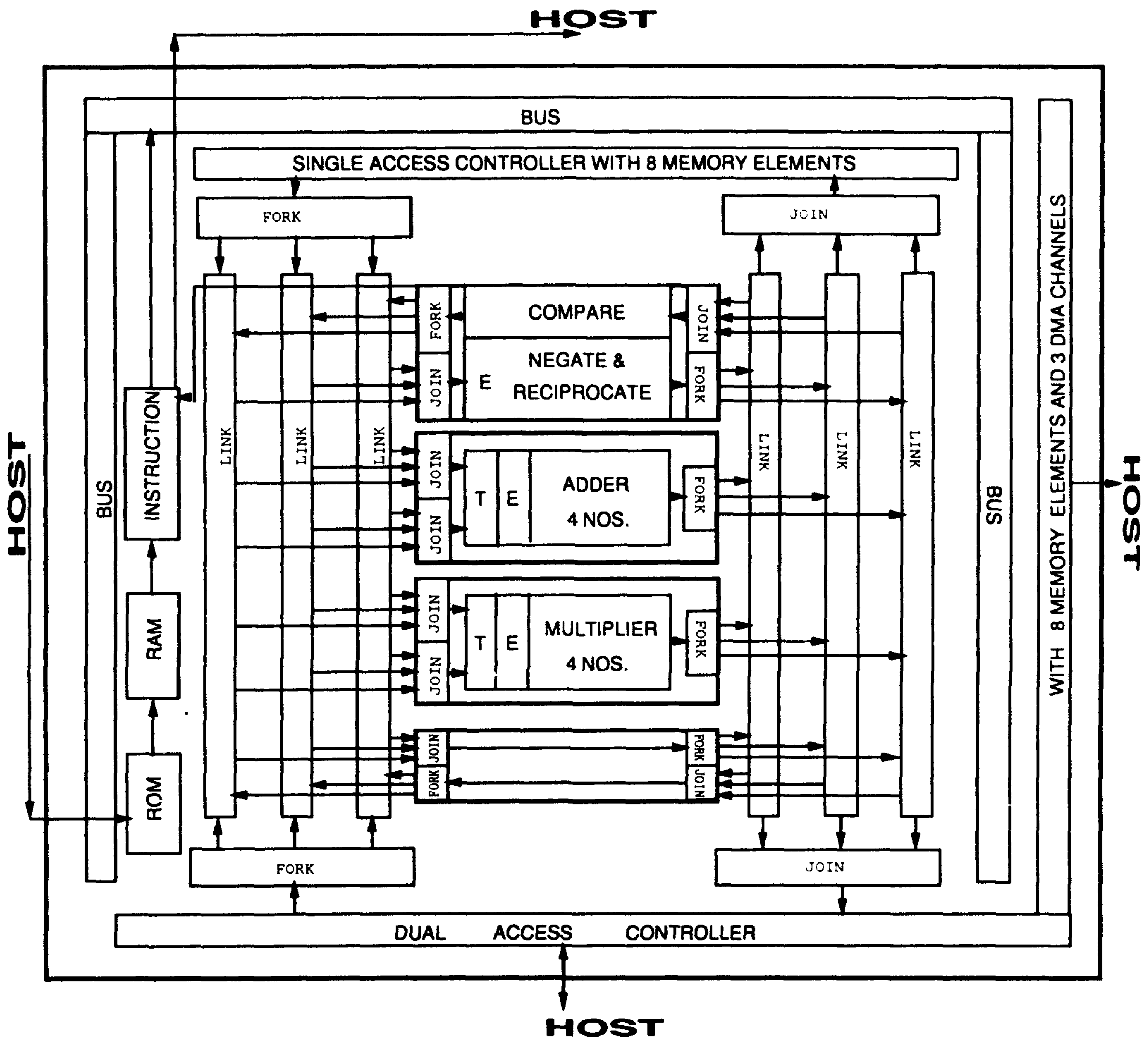


Fig. 5. Layout of MCAP on an MCM

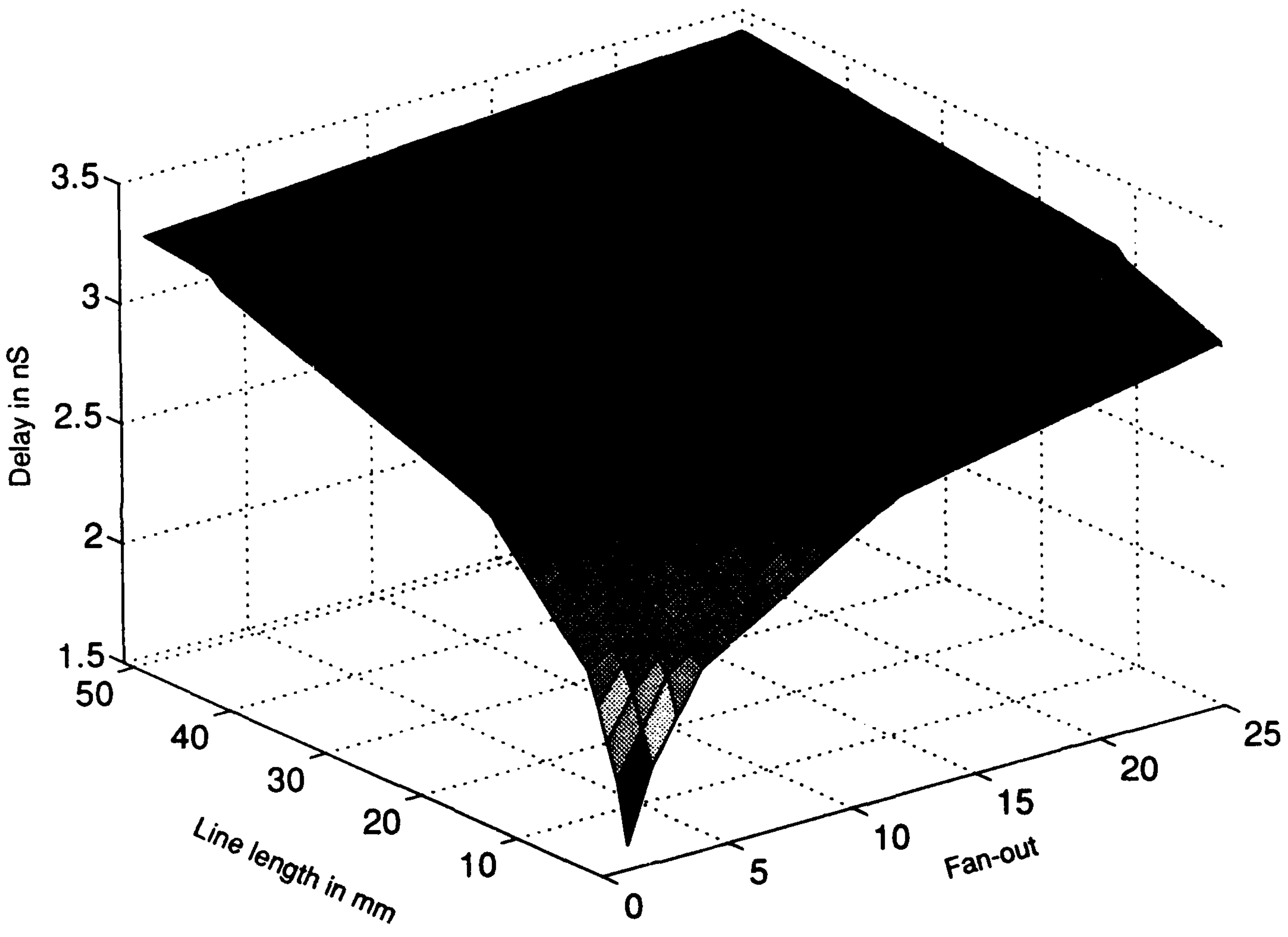


Fig. 6. Buffer and interconnect delay

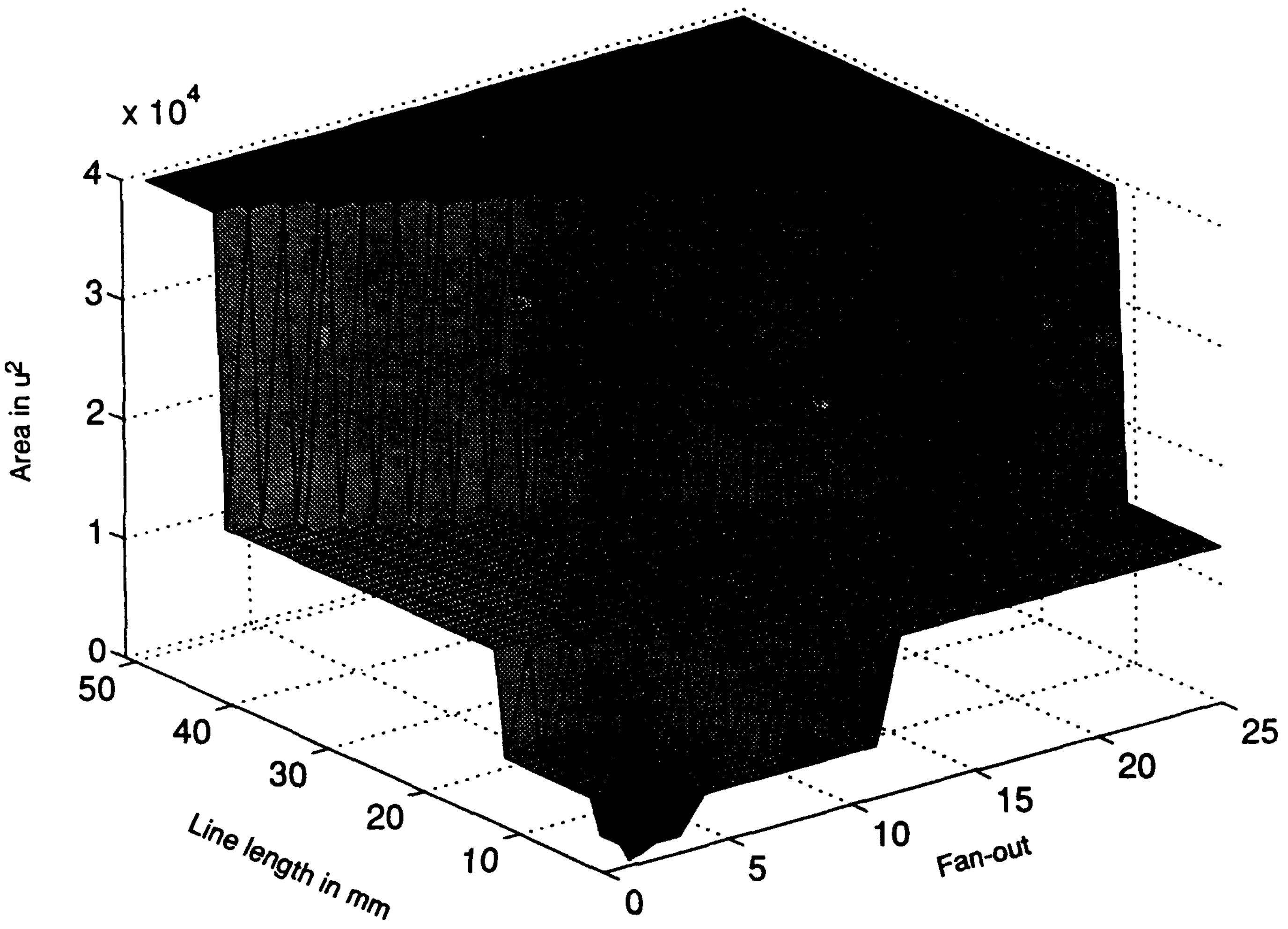


Fig. 7. Buffer area

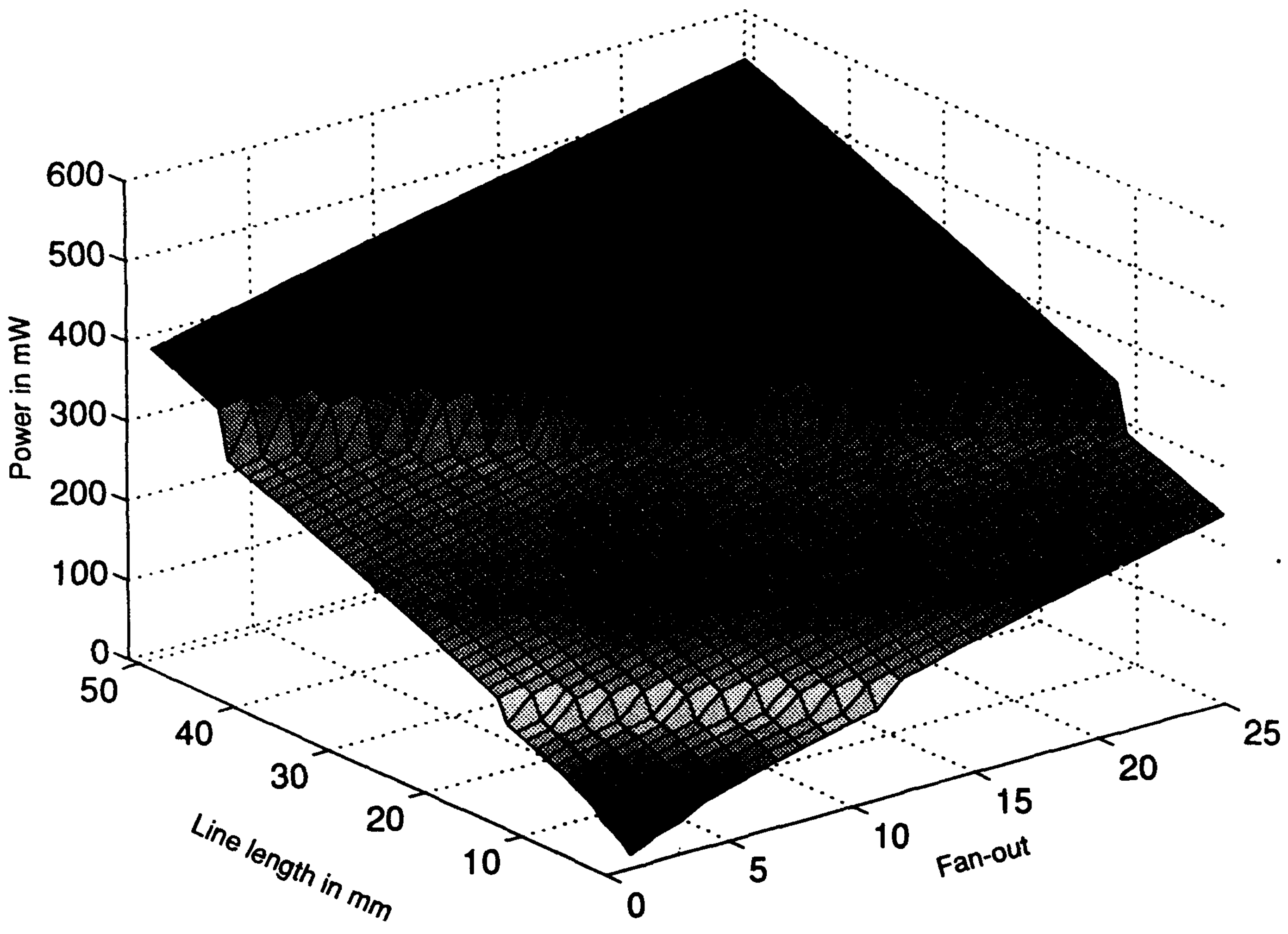


Fig. 8. Power dissipated in the Buffer

Att: B

APPLICATION ALGORITHMS FOR A MODULARLY CONFIGURABLE ATTACHED PROCESSOR[1]

Glenn Gibson, Yi-Chieh Chang, Sing-Wai Wu, Brito Alejandro
Yu-cheng Liu, Sergio Cabrera, and Vijay Singh
Electrical Engineering Department
The University of Texas at El Paso
El Paso, Texas 79968-0523

ABSTRACT

A new architecture for high-performance parallel attached processors is described in this paper. Based on this architecture, an attached processor can be implemented as multiple memory-to-memory pipelines, each being constructed with a class of fundamental components. The unique features are that the attached processor can be configured to match a set of algorithms and its memory controllers can be programmed to fit the access patterns required by the algorithms. As a result, high utilization of the processing logic for given sets of algorithms can be obtained. Detailed performance analyses of application algorithms, including solution of linear equations and fast Fourier transform, based on the proposed architecture are given. Technology to implement the proposed architecture and packaging considerations are also discussed.

Index Terms: Attached processor, interconnected pipeline, memory-to-memory pipeline, sustained execution rate.

[1]The work reported in this paper was supported in part by the Office of Naval Research under Grant No. N00014-93-1-1343. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of the funding agencies.

1 Introduction

An attached, or back-end, processor is a processing system that is connected to a host computer for the purpose of very quickly executing most of the overall system's computational tasks. In such an organization, "the host is a program manager which handles all I/O, code compiling, and operating system functions, while the back-end attached processor concentrates on arithmetic computation with data supplied by the host machine" [1].

Typical early attached processors were the AP-120B and FPS-164 made by Floating Point Systems, Inc., the IBM 3838, and the MATP made by Datawest, Inc. [1], [2], [3]. These attached processors all have their own data memories and transfer data between these memories and the main memories of their hosts using DMA data channels. They also include their own code memories where subprograms may be permanently stored or downloaded from their hosts. These subprograms are initiated by commands from the host and supervise the data flows from the attached processor's data memories, through the attached processor's processing elements, and back into the data memories.

Although the early attached processors included limited multiprocessing, the more recently implemented processing arrays are also controlled by a host (e.g., the PAX computer [4]) and are designed to perform most of the overall system's computational tasks. Therefore, these arrays and even the array processing portions of today's supercomputers, such as the Cray series [1], [3] could be interpreted as attached processors, although the host is then sometimes referred to as a front-end computer.

The specific purpose of an attached processor is to execute members of a set of algorithms very quickly. The broader the set of algorithms the more generally applicable the attached processor. The underlying goal of the designer is to efficiently utilize the hardware for as broad a set of algorithms as possible. However, for most current designs, the average sustainable execution rates have been found to be only 5% to 20% of their peak rates, which are determined by summing the maximum computational rates of the processing elements. For example, the sustainable rate for a Cray X-MP with four processors may be as low as 5% for some algorithms [5]. Also extensive evaluations of recent high-performance computations using Lapack are given in [6] and using NSA parallel benchmarks are given in [7]. Although some of the lost efficiency is necessitated by the algorithms, much of it is due to memory accessing and contention for shared resources in general, including internal buses.

Described in this paper is a class of high-performance attached processors called Modularly Configurable Attached Processors (MCAPs) which can attain quickness and high utilization through:

- Closely matching their architectures to the set of algorithms they are to execute.
- Overlapping of processing and memory accessing by using memory prefetching.
- Minimizing the movement of data.
- Using a high-speed technology with MCM or wafer scale implementations.

An MCAP is constructed from the component types specified in Sec. 2. These component types are such that each member of the class may include parallel processing, memory-to-memory pipelines, and be constructed in a building block fashion. They encompass routing components (including buses) as well as memory, control, and processing components. By overlapping processing with memory accessing and matching an architecture with a set of algorithms, it is predicted that the average sustainable rate for a specific set of algorithms can attain at least 60% of the peak rate. By defining components that are simple enough to be fabricated onto single low-density ICs, a high-speed technology may be used.

Much of an MCAP's efficiency is gained by distributing the instructions for the next algorithm (or algorithm phase) to the various components while the current algorithm (or phase) is executing. Once the algorithm begins, these instructions dictate the modes, routing patterns, prefetching patterns, and so on of the components receiving them. After an algorithm starts, each component operates more or less on its own except for responding to its handshaking signals. Efficiency is further enhanced by prefetching operands from the memory subsystems. Prefetching using programmed patterns avoids the misses that result from using ordinary caches.

Section 2 gives an overview of the MCAP architecture and the fundamental components required to construct an MCAP. One of the most desirable features of the MCAP is its ability to be modularly configured to match given sets of algorithms in order to predetermine the data patterns to be prefetched. Sections 3 and 4 illustrate how to match typical application algorithms with MCAP architectures. Detailed performance analyses are given, which show high sustainable rates relative to the peak performance. The algorithms analyzed include solution of linear equations and fast Fourier transforms. Finally, implementation considerations of the MCAP including semiconductor technoligies and packaging are discussed in Section 5.

2 MCAP Architecture

An MCAP is an attached processor that is constructed entirely from a standard set of connections and components. This standard set consists of three types of asynchronous connections and twelve types of components. The definitions of the connection and component types provide a standard set of rules that allow the components to be easily configured in different ways to construct attached processors that can efficiently perform different sets of algorithms.

An MCAP has exactly one instruction component and it is connected to a memory component for storing instructions. Most of this memory component is a ROM that contains the subprograms needed to execute the algorithms, but some of it is a RAM that can receive instructions (those that initiate the subprograms) from the host.

An MCAP operates by drawing an instruction stream from the memory component into the instruction component. The instruction component uses internal instructions in the stream to form external instructions that are then distributed to the other non-memory components through the MCAP's bus component. All components in the instruction stream include input instruction queues. When the non-memory components have received all of the instructions needed to perform an algorithm, they automatically prefetch the data from the memory components, route the data to and from the processor components and store the results back into the memory components. All non-memory components have input data queues. DMA units built into some controller components, which are the components that supervise all memory accessing, are used to automatically transfer data between the host's main memory and the MCAP's memory components while the algorithm is executing. Also, the instruction and data streams are separate, thereby allowing the instructions needed for the next algorithm to be distributed while the current algorithm is executing.

The three types of connections are referred to as memory, instruction, and data connections. All connections are asynchronous and, therefore, must include handshaking lines as well as data and, perhaps, address lines. Each memory component is connected to its controller component by a single memory connection that consists of a bidirectional data bus, a unidirectional address bus and a Request (Req)/ Acknowledge (Ack)/Memory Request (MReq) handshaking triplet. Instruction connections are for passing instructions from the instruction component to the bus component and from the bus component to one of the other non-memory components. An instruction connection consists of unidirectional instruction and address buses and a Req/Ack handshaking pair. The component that is to receive the instruction is indicated by the a component number on the address bus. A transfer is initiated when the sending component puts an address on the address bus, an instruction on the instruction bus and begins the handshaking. The Free line provides the instruction component with a means for detecting when all components connected to the bus component are completely inactive. Except for the connections to memory components, all connections used to transfer data are data connections. They are used to pass data to and from the processors through routing components and consist of only a unidirectional data bus and a Req/Ack pair. A data transfer is consists of placing data on the data bus and initiating the handshaking. Except for a write to a memory component, all transfers include the latching of an instruction or datum into a queue at the receiving end.

The twelve types of components are divided into six categories. As mentioned earlier, an MCAP contains one memory component for storing instructions, one instruction component for executing internal instructions and forming external instructions, and one bus component for distributing the instructions. In addition, an MCAP may contain several controller, router, and processor components and several other memory components for storing data. Each non-memory component that is used during the execution of an algorithm contains an instruction input queue, one or more data input queues, and control logic that includes a number of registers. The instructions for an algorithm received by a component fill these registers and then the register contents dictate the activity within the component while the algorithm is executed. Each of these components contain a Number of Operands Output (NumOpsOut) register that is always the last register filled before the component begins its part in the execution of the algorithm. Each time the component outputs an operand, the NumOpsOut register is decremented. When the NumOpsOut register becomes zero, the component has completed its part in executing the current algorithm. It may then distribute new values, those needed for the next algorithm, from its instruction input queue to its registers. This cycle may continue indefinitely. Except for reacting to the handshaking signals in its connections, each component acts independently.

The processor components are used for performing unary and binary arithmetic/logic operations. There are three types of processor components. There are one-input elementary components, two-input components and comparator components. These components contain only two registers, a mode register and a NumOpsOut register. The mode register dictates the actions taken by the component and the NumOpsOut register gives the total number of operands that is to be output before the current algorithm is completed. Both the elementary and two-input components may be used for either unary or binary operations, depending on the mode. A comparator component is used to compare values or find maxima or minima and returns appropriate flag values to the instruction component.

Routing components are for directing data along the proper paths. There are three types of routing components, join components with multiple input and one output, fork components with one input and multiple output, and link components with more than one input and more than one output. In addition to the mode and NumOpsOut registers, they contain registers for dictating their input and output patterns while the current algorithm is being executed. Fork and link components may include broadcasting in their output patterns.

There are three types of controller components, RAM components, single-access components, and dual-access components. All controller components are for prefetching operands from and storing results in their associated memory components. A single-access component differs from a RAM component by permitting its associated memory to be divided into partitions. In addition to the logic in a single-access component, a dual-access component contains DMA units and connections for communicating with the host's main memory. All controller components have an output data connection for outputting operands to the remainder of the MCAP and an input data connection for inputting results from the MCAP.

An example architecture is given in Fig. 1. Its processing subsection includes a comparator, a negator (elementary component), a reciprocator (elementary), a set of pipelined adders capable of accumulation, and a set of pipelined multipliers. Each adder or multiplier is constructed of two or more stages (a two-input component followed by one or more elementary components). All communications to and from the processing components are through six link components, three on each side of the processor. Join and fork components are provided to allow flexible use of the link components. Also, to allow for accumulation there is a feedback connection between the fork component at the output from each adder and the join component at the input to the adder. There is a dual-access component to provide intermediate memory and a connection to main memory. The single-access component provides internal storage.

3 Performance analysis of an MCAP

One of the major advantages of the MCAP architecture compared to cache memory or vector processors is the ability of prefetching a pre-determined data pattern for any given algorithm. The controller component can be set up in a way that the data stream for a given algorithm can be prefetched in ahead of time, so that the processor components can be kept busy during the entire operation period. Thus, the sustained speed of an MCAP can be at least at 60% of the peak performance for most of the application algorithms.

To demonstrate the concept of prefetch data pattern in an MCAP, a sample architecture given in Fig. 1 is considered first. All processing components are made up of four stages each having a stage delay time of 20 ns. This gives a peak performance of 450 MFLOPS. Six 10 ns link components are used to serve the processing components, three at the inputs and three at the outputs. One link component at the input and output is reserved for feedback operation, i.e., to feedback output from an adder (or multiplier) to another multiplier (or adder). So, only two out of the three link components at the input or output can bring in data from memory component or store data to the memory component.

This allows 300 M transfers/s at both the inputs and outputs, but only 200 M transfers/s to/from memory subsystem. Because the fork and join components are each connected to two link components they have been given a delay time of 5 ns. The FIFO buffers that are connected to the dual controller component also have a 5 ns per stage delay time. To supply 200 M operands/s the memory subsystem of the dual controller consists of 8 interleaved 40 ns memory components.

To demonstrate the effectiveness of such a system consider the solution of a system of linear equations using Gaussian elimination. A group of n linear equations can be expressed as follows:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= C_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= C_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= C_n. \end{aligned} \tag{3.1}$$

Gaussian elimination is one of the most efficient methods to solve Eq. (3.1). In the first phase of the solution Eq. (3.1) is reduced to

$$\begin{aligned} a'_{11}X_1 + a'_{12}x_2 + \cdots + a'_{1n}x_n &= C'_1 \\ a'_{22}x_2 + \cdots + a'_{2n}x_n &= C'_2 \\ &\vdots \\ a'_{nn}x_n &= C'_n \end{aligned} \tag{3.2}$$

In the second phase, the variables x_1 to x_n can be obtained by the following computations:

$$\begin{aligned} x_n &= \frac{C'_n}{a'_{nn}} \\ x_k &= \frac{C'_k - \sum_{i=k+1}^{n} a'_{ki}x_i}{a'_{kk}}, \quad k = n-1, n-2 \ldots, 1 \end{aligned} \tag{3.3}$$

The total number of computations needed to solve Eq. (3.1) is derived below. The following algorithm reduces Eq. (3.1) to Eq. (3.2):

Algorithm 1

```
for $k = 1$ to $n - 1$ do
    for $i = k + 1$ to $n$ do
        for $j = k + 1$ to $n$ do
            $a'_{ij} \longleftarrow a_{ij} \times a_{kk} - a_{ik} \times a_{kj}$
            $j \longleftarrow j + 1$
        end_do
        $C'_i = C_i \times a_{kk} - C_2 \times a_{ik}$
        $i \longleftarrow i + 1$
    end_do
    $k \longleftarrow k + 1$
end_do
```

The total number of computations in Algorithm 1 is

$$3[n(n-1) + (n-1)(n-2) + \cdots + 2] = \frac{1}{2}n(n-1)(2n-1) + \frac{3}{2}n(n-1). \tag{3.4}$$

From Eq. (3.2), x_1 to x_n can be calculated using Eq. (3.3). The number of computations in Eq. (3.3) is

$$2 + 4 + \cdots + 2n = \sum_{i=1}^{n}(2i-1) = n(n+1). \tag{3.5}$$

The total number of computations required to solve n simultaneous linear equations is the sum of Eqs. (3.4) and (3.5), which is

$$C = \frac{1}{2}n(n-1)(2n-1) + \frac{3}{2}n(n-1) + n^2 = \frac{1}{2}n(n-1)(2n-1) + \frac{n}{2}(5n-1) \tag{3.6}$$

The detailed data flow and computing pipelines in the MCAP can be illustrated using the diagram in Table. 1. The memory controller is set up to feed in a pair of a_{ij}'s for the multipliers and the outputs of the multipliers are chained to the adders using the L components. The multiplications $a_{ij} \times a_{kk}$ and $a_{ik} \times a_{kj}$ are executed in different multipliers. For example, the first pair of multiplications $a_{ij} \times a_{kk}$ and $a_{ik} \times a_{kj}$ is executed in M_1 and M_2, respectively. The second pair of multiplications $a_{ik} \times a_{kj}$ and $a_{ik} \times a_{kj}$ is executed in M_3 and M_4, respectively. This sequence will repeat until all multipliers are busy executing. Note that a_{kk} can be kept in the M component for the two inner loops and a_{ik} can be kept in the M component for the innermost loop. Therefore, the D controller does not need to feed a_{kk} and a_{ik} into the multipliers with each multiplication. The data streams to each of the multipliers is shown below. The following tables assume $n = 20$, i.e., there are 20 simultaneous linear equations.

14	13	12	11	10	9	8	7	6	5	4	3	2	1	
—	$a_{2,12}$	—	$a_{2,10}$	—	a_{28}	—	a_{26}	—	a_{24}	—	a_{22}	—	a_{11}	$\longrightarrow M_1$
$a_{1,12}$	—	$a_{1,10}$	—	a_{18}	—	a_{16}	—	a_{14}	—	a_{12}	—	—	a_{21}	$\longrightarrow M_2$
—	$a_{2,13}$	—	$a_{2,11}$	—	a_{29}	—	a_{27}	—	a_{25}	—	a_{23}	—	a_{11}	$\longrightarrow M_3$
$a_{1,13}$	—	$a_{1,11}$	—	a_{19}	—	a_{17}	—	a_{15}	—	a_{13}	—	—	a_{21}	$\longrightarrow M_4$
Stage 2 (2 × 10 ns)		**Stage 1 (2 × 10 ns)**		**Stage 4 (2 × 10 ns)**		**Stage 3 (2 × 10 ns)**		**Stage 2 (2 × 10 ns)**		**Stage 1 (2 × 10 ns)**				
Multipliers produce outputs														

Table 1. Data pattern to fill the multipliers

Each clock cycle in Table. 1 is 10 ns, for every 2 clock cycles the multiplier can move the operands to the next stage while keeping either a_{kk} or a_{ik} in the first stage. Thus, each multiplier only needs one operand for each multiplication after the 3^{th} clock cycle. At the end of the 10^{th} clock cycle, the pipelines in all 4 multipliers will be filled and two products are produced every 10 ns. These outputs are chained to the adders to perform the subtractions. Although M_1 and M_3 (or M_2 and M_4) can produce one product at the same clock cycle, there is only one link component at the input and output that can be used to feedback the results to the adders. So, only one product can be sent to the adders at one clock cycle. The data pattern is shown in Table. 2. In this table, pairs of outputs from the multipliers are sent to the adders where they are subtracted. Since each adder also consists of a 20 ns per stage pipeline, an adder can be filled with new operands every 2 clock cycles. Because A_1 can start to execute its two operands at the 12^{th} clock cycle, it can take two more operands at the 14^{th} clock cycle. So, only two adders are needed to perform the subtractions of the outputs generated by the 4 multipliers.

At the end of the innermost loop, a_{ik} needs to be updated in the multipliers that hold this coefficient, but the other multipliers that hold the a_{kk} can continue to execute. However, in order to match the output patterns of the 8 multipliers, an idle cycle is inserted into those multipliers that hold the a_{kk}. The data pattern is shown in Table. 3.

At the end of the two inner loops, two extra operands need to be sent to the multipliers. Also, two extra clock cycles need to be inserted at the beginning of the outer loop. The data pattern is shown in Table. 4.

22	21	20	19	18	17	16	15	14	13	12	11	
—	—	—	—	—	—	—	—	—	—	—	M_1	$\longrightarrow A_1$
—	—	—	—	—	—	—	—	—	—	M_3	—	$\longrightarrow A_1$
—	—	—	—	—	—	—	—	—	M_2	—	—	$\longrightarrow A_2$
—	—	—	—	—	—	—	—	M_4	—	—	—	$\longrightarrow A_2$
—	—	—	—	—	—	—	M_1	—	—	—	—	$\longrightarrow A_1$
—	—	—	—	—	—	M_3	—	—	—	—	—	$\longrightarrow A_1$
—	—	—	—	—	M_2	—	—	—	—	—	—	$\longrightarrow A_2$
—	—	—	—	M_4	—	—	—	—	—	—	—	$\longrightarrow A_2$
—	—	—	M_1	—	—	—	—	—	—	—	—	$\longrightarrow A_1$
—	—	M_3	—	—	—	—	—	—	—	—	—	$\longrightarrow A_1$
—	M_2	—	—	—	—	—	—	—	—	—	—	$\longrightarrow A_2$
M_4	—	—	—	—	—	—	—	—	—	—	—	$\longrightarrow A_2$
Stage 2 (2 × 10 ns)		Stage 1 (2 × 10 ns)		Stage 4 (2 × 10 ns)		Stage 3 (2 × 10 ns)		Stage 2 (2 × 10 ns)		Stage 1 (2 × 10 ns)		

Adders produce outputs

Table 2. Data pattern to fill the adders for subtraction

—	a_{36}	—	a_{34}	—	a_{32}	—	—	$a_{2,20}$	—	$a_{2,18}$	$\longrightarrow M_1$
a_{16}	—	a_{14}	—	a_{12}	—	a_{31}	$a_{1,20}$	—	$a_{1,18}$	—	$\longrightarrow M_2$
—	a_{37}	—	a_{35}	—	a_{33}	—	—	C_2	—	$a_{2,19}$	$\longrightarrow M_3$
a_{17}	—	a_{15}	—	a_{13}	—	a_{31}	C_1	—	$a_{1,19}$	—	$\longrightarrow M_4$

Beginning of another inner most loop ($i = 3$) End of the first inner most loop

Table 3. Data pattern at the end of the innermost loop

—	a_{37}	—	a_{35}	—	a_{33}	—	a_{22}	—	$a_{20,20}$	—	$a_{20,18}$	$\longrightarrow M_1$
a_{27}	—	a_{25}	—	a_{23}	—	a_{32}	—	$a_{1,20}$	—	$a_{1,18}$	—	$\longrightarrow M_2$
—	a_{38}	—	a_{36}	—	a_{34}	—	a_{22}	—	C_{20}	—	$a_{20,19}$	$\longrightarrow M_3$
a_{28}	—	a_{26}	—	a_{24}	—	a_{32}	—	C_1	—	$a_{1,19}$	—	$\longrightarrow M_4$

Beginning of the second outer loop (k=2) End of the $(n-1)^{th}$ inner most loop

Table 4. Data pattern at the end of the two inner loops and the start of the second outer loop.

The total number of clock cycles needed to feed in all the data patterns in Algorithm 1 is:

$$\begin{aligned} N_{cycle1} &= \sum_{i=1}^{n-1} i^2 + 3\sum_{i=1}^{n-1} i + 2(n-1) = \frac{1}{6}n(n-1)(2n-1) + \frac{3}{2}n(n-1) + 2(n-1) \\ &= \frac{1}{6}n(n-1)(2n-1) + \frac{1}{2}(n-1)(3n+4). \end{aligned} \tag{3.7}$$

The sustained performance in executing Eq. (3.3) is not as good as for Algorithm 1, because x_k must be calculated sequentially. Basically, the reciprocal of $1/a'_{nn}$, multiplication of $a'_{ki}x_i$, and the subtraction of $C'_k - a'_{ki}x_i$ must be executed in series. So, when $n = 1$, 8 pipeline stages (16 clock cycles) are needed to calculate x_1, and 12 pipeline stages are needed when $n = 2$. When $n \geq 3$, the sequence of multiplications and subtractions can be executed in pipeline fashion, so only one extra clock cycle is needed for each extra multiplication or subtraction. Thus, we have the following equation.

$$N_{cycle2} = \begin{cases} 16 & n = 1 \\ 40 & n = 2 \\ 40 + 24(n-2) + 2\sum_{i=1}^{n-2} i & \\ = 40 + (n-2)(n+23) & n \geq 3 \end{cases} \tag{3.8}$$

For $n \geq 3$ the sustained performance for solving Eq. (3.1) is equal to $C/(N_{cycle1} + N_{cycle2})$.

$$\begin{aligned} T_n &= (N_{cycle1} + N_{cycle2}) \times 10\,ns \\ MFLOPS_{sustained} &= \frac{\frac{1}{2}n(n-1)(2n-1) + \frac{n}{2}(5n-1)}{T_n} \end{aligned} \tag{3.9}$$

For example, when $n = 20$, we have $T_n = 38.92\mu s$ and the total number of floating point operations is $C = 8400$, thus the sustained performance for executing Eq. (3.6) is $C/T_n \approx 215.8$ MFLOPS. The theoretical peak performance of the MCAP shown in Fig. 1 is 450 MFLOPS. Thus, the sustained performance of the MCAP is 48% of its peak performance. When $n = 100$, the sustained performance is 284.4 MFLOPS, which is 63.1% of the peak performance. When $n = 1000$, the sustained performance is 298.5 MFLOPS, which is 66.5% of the peak performance. The sustained computation speed and ratio to the theoretical peak performance vs. n is plotted in Fig. 2. Note that in this figure two ratios are given, one is for configuration shown in Fig. 1, the other ratio is given for the case of 4 multipliers, 2 adders, and 1 reciprocal, because only 2 adders are needed to solve the simultaneous linear equations.

The advantage of the MCAP over a conventional vector processor, such as Cray-1, is that the data flow is supported by an intelligent memory controller which knows the pattern of the data flow for a given problem. Therefore, the processing components can be kept busy most of the time as long as the data transfer rate on the memory controller and the L components is fast enough. In the Cray-1 computer, the data must first be stored in vector registers. For small problem sizes, such as $n < 20$, the sustained performance is usually less than 10% of its theoretical peak performance. However, in the MCAP, the sustained performance is around 48% (62%) of its peak performance even when $n = 20$.

4 Matrix Multiplication in Discrete Fourier Transform Evaluation

In signal processing, frequency analysis is very often used to get a better knowledge of the nature of the signal. The Discrete Fourier Transform (DFT) used to obtain the spectrum of an N-point sequence is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n)\exp(-j2\pi nk/N). \tag{4.1}$$

Eq. (4.1) can be formulated as a matrix-vector product; the matrix is formed by the N coefficients obtained from the different powers of the N-th root of unity and the vector is formed from the N data samples.

The total number of arithmetic operations for an N-point DFT is N(N-1) multiplications and N(N-1) additions. If we consider complex signals, there will be 4N(N-1) real multiplications and 4N(N-1) real additions. A total of approximately $8N^2$ arithmetic operations.

The basic architecture, shown in Fig. 1, used in performing matrix multiplication, can be used to perform the DFT evaluation. In order to obtain a faster DFT, we may apply the *divide and conquer* strategy which is simply to break a large matrix-vector multiplication into many smaller ones.

Effectively, there is no saving in the computational effort. The benefit of this approach is that we can obtain the DFT at a faster rate by sharing the computation load among the several processors elements.

An efficient algorithm to evaluate the DFT is the Fast Fourier Transform (FFT), which has two versions: Decimation-in-time (DIT) and Decimation-in-frequency (DIF). This is another approach to obtain a faster DFT by reducing the computational requirements. It is believed that a combination of the two approaches, that is sharing a reduced number of computations, will be a good strategy to obtain the DFT in a short time. In this paper, we will concentrate on the study of the efficient implementation of the one-stage FFT algorithm using an MCAP architecture. The division of a long DFT sequence will be considered later. Following, we study the implementation of one-stage FFT algorithm using the basic MCAP architecture, and another architecture is proposed for an efficient FFT evaluation.

4.1 Comparison of the DIT and the DIF FFT Algorithms

It is well-known that by using the FFT algorithm, the number of arithmetic operations can be reduced drastically to the order of $2N\log_2N$ real multiplications and $3N\log_2N$ real additions for an N-point DFT. Depending on the fashion of separating the input data, there are two approaches to the FFT algorithm. When the input data are decomposed into even or odd, the FFT is achieved from *decimation in frequency* (DIF). When the input data is processed in first and second half, then we are evaluating the FFT by *decimation in time* (DIT). In each case, further division of the partial results is possible in the same manner. Fig. 3 shows the basic butterfly computations in both the DIF and the DIT FFT algorithms respectively. Fig. 3 also shows the butterflies with the real and imaginary parts separately. These last figures give a better description of the algorithms. Repeating the DIF and DIT operations to intermediate results, the flow graph of a 16-point DIF- and DIT- FFT can be obtained as shown in Fig. 4 [DeFatta].

It can be seen that the DIF-FFT and the DIT-FFT are working in opposite manner. The DIF-FFT is working in a *decentralizing* way in which the processing can be operated independently in the following stage. However, no data processing is possible before the previous processor has completed its work. On the other hand, the DIT-FFT is working in a more and more *centralizing* manner. At the input stage, we may have many basic two-input butterflies. As we are proceeding to the final stage, we will have larger distance between the input of the butterflies. This means that if parallel operations for the DIF-FFT and the DIT-FFT are introduced, they are occurring in different order. Parallelism has to be introduced at the later stages in the DIF-FFT algorithm, but at the early stages in the DIT-FFT counterpart.

A closer inspection at the basic butterfly units tells us that in the DIT-FFT approach, the multiplication is done first and in the DIF-FFT case, the multiplication is done at an intermediate stage. Furthermore, by referring to the computation expansion tree for the DIF-FFT and the DIT-FFT in Fig. 5, it can be seen that the DIT approach has a smaller span and less number of stages than the DIF approach. Therefore, in our further study, we consider only the DIT approach because it is more suitable for parallel implementation and it involves less computations.

4.2 The FFT Implementation with the Basic Architecture

As discussed in the previous section, several features of the DIT-FFT favor its use in the FFT implementation. On the other hand, the DIT-FFT is easier to adapt to our architecture. Since the basic architecture is aimed to perform general algorithms, some of its components are not used in the FFT algorithm. A data-flow timing diagram showing the processing elements (PEs) that are occupied at a certain stage is constructed to help analyzing if a linear pipeline can be achieved.

Fig. 6 shows the data-flow timing diagram for the DIT-FFT implementation using the basic architecture. The total time required for one-stage FFT computation is 400ns, from memory to memory. It should be noted that several PEs are reused in just one FFT computation. This feature hinders our objective of achieving a linear pipeline since the more reuse of a PE in a computation, the more PE time has to be reserved for the feedback of partial results. Consequently, the processing new data will be delayed. Therefore, the input data cannot flow smoothly through all the PEs. In the timing estimation, we have assumed that the delays for each architecture's element are as follows: Memories, links : 10ns, Joints, forks : 5ns, and Adders and Multipliers : 20ns per stage.

The highest degree of parallelism is 6 computations, and this occurs at the 25-30ns time-slot. A high degree of contention occurs with the links because many partial results are fed back from the multipliers' and the adders' outputs to the adder inputs for further processing. The minimum time-lapse for the next FFT computation is approximately 290ns. To solve this problem, we propose another architecture in Fig. 7, that minimizes this type of data feedback. In addition, several elements (the negator, the reciprocator) that are not used in the DIT-FFT algorithm are changed by adders in order to achieve a more linear data flow.

4.3 The FFT Implementation with the Proposed Architecture

The proposed architecture is conceived based on the computation expansion tree in Fig. 5. First, the multiplications are performed; next, there are a substraction and an addition to get intermediate values that finally arrive to the last level, where two additions and two substraction are executed. Once the described flow is figured out, the modifications to the basic architecture follow. The major modifications in the new proposed architecture are that the negator and the reciprocator are replaced by two adders. The multipliers' outputs are hardwired to the inputs of two adders which are reversed. The outputs of these two adders are hardwired to the forks inputs connected to the remaining adders. In this way, we do not have to use the links (#0, #1 or #2) to feed back the partial results anymore. This architecture also relieves us from re-using two adders due to replacing of the unused elements with adders. The data-flow timing diagram for the DIT-FFT implementation is shown in Fig. 8. It can be seen that the flow is much linear than the one for the original architecture. The highest degree of parallelism in one butterfly operation is 5 computations and it occurs at the 15ns time-slot. Clearly, when several butterfly operations are in the pipeline the degree of parallelism goes from 10 to 15 computations as observed in Fig. 9. Using the same delays for the elements as before, the total delay for one-stage FFT computation is 350ns (from memory to memory). The minimum time-lapse fo- the next FFT computation is 5ns. For the first stages of computations, the a's operands are loaded in the single access component. After the first stages, the next ordering of data will be done by the links' components.

4.4 Conclusions and Future Work

It is found that the general matrix-vector/matrix-matrix product is different from the FFT algorithm in that the adder output is continually updated with the next input in the matrix multiplication. This is not the case for the FFT algorithm, and the direct feedback path in the two adders are not helping very much. This is the reason why we removed the two direct feedback paths in the two adders. The study shows that the basic architecture is not the best in performing the FFT algorithm. However, with the suitable choice of the FFT approach and modifications of the basic architecture, we are able to obtain a relatively linear pipeline. In addition, the new architecture allows the next FFT computation to take place about $3\frac{1}{2}$ times faster than the basic architecture with almost 90% efficiency in the components.

The modifications made in the basic architecture are easy. There is no additional component requirement. In fact, some of the components present in the basic architecture are redundant for the FFT algorithm. Overall, the proposed architecture can attain a more efficient FFT calculation without a great cost for modifications.

The future work is related to the simulation of the discussed algorithms using the presented architectures. To achieve that, the MCAP simulator, assembler and editor will be used. First, the basic architecture and the proposed architecture must be created using the MCAP editor which will produce files with the information of each component and their connections. Next, a program of the DIT-FFT algorithm must be implemented using the basic architecture and another using the proposed architecture. Then, the programs must be fed to the MCAP assembler in addition to the architecture files created by the editor. The assembler will create a load file based on the program file and the corresponding architecture file. Finally, the load file and the architecture file will be fed to the simulator that will simulate the DIT-FFT algorithm using the basic or the proposed architecture. The simulator's results will be time analyses and efficiency information per component among other information. The simulation could be done several times using different parameter values for the architectures' components until the best set of values and the best architecture distribution is found. Then, the design and implementation stage of the components could start. [8].

5 Technology and packaging considerations

The most important parameters related to the design of high performance computers are speed, power consumption, gate capacity of a single chip, yield, wafer size, and production cost. The highest possible speed is the ultimate goal, but this goal is tempered by physical and economical limitations such as heat dissipation and yield. The current leading contenders for implementing high-performance computers are silicon based bipolar complementary metal-oxide-semiconductor (BiCMOS) logic, silicon based emitter coupled logic (ECL), and GaAs technologies [9].

Of these technologies, ECL offers high speed, but its relatively high power dissipation may well prove prohibitive for some applications. BiCMOS provides lower power dissipation and the largest number of gates per chip; but it is relatively slow. Because of its high speed and low power dissipation, the GaAs technology is clearly a best technology for high speed computer applications. Higher than 500 MHz operation and 88 nW/MHz gate power dissipation are now available. Operation above 750 MHz with an extremely low power dissipation of 44 nW/MHz gate have been demonstrated [10]. Since GaAs is a relatively new technology, it still suffers from small wafer size (10 cm diameter), low yields and high production costs. It is expected however that these problems will be overcome in the near future as the technology matures.

With regard to packaging, speed is primarily determined by communication delays. These delays can be reduced by increasing the power to the drivers, but this increases the heat that must be dissipated from the system's ICs. There are four communication levels corresponding to the four packaging levels:(1) on chip, (2) on-package but off-chip, (3) off-package, and (4) off-PC-board. In going from on-chip to off-PC-board the communication paths increase in length and there is a drop off in communication speed. For a given technology one would put as many gates on each chip as yield limitations would permit. Beyond this, the current trend in high-performance design is toward using on-package, off-chip multichip modules (MCMs). Although MCMs are expensive to produce they provide a much improved speed-power tradeoff. Experimental work has been done toward 3-dimensional MCMs [11], which may increase inter-chip communication rates to 1 GHz, as opposed to 250 MHz for an equivalent 2-dimensional chip array. MCAPs are constructed of relatively simple components so that gate capacity is not a limiting factor. However, the communication paths would be at least one operand (say 64 bits) wide and must be very fast. Therefore, heat dissipation by the driver circuits would be a serious problem.

The example MCAP architecture (Fig. 1) was evaluated for MCM implementation [12]. The layout for the MCM implementation is shown in Fig. 10. Because of its commercial maturity, CMOS technology was picked as the first (benchmark) technology to be evaluated. Transistor count for implementing the MCAP was estimated at 9.85 million. In the proposed architecture, the bottleneck is the communication through the LINK elements because of their high fan-out and relatively large interconnection distances. This means that the output buffers for these elements must be relatively large. For the output buffers, delay power and area calculations were made as functions of *fan-out* (F) and *interconnection length* [12]. For example, a LINK element with a fan-out of 19 and an average interconnection length of 2 cm has a load capacitance of 34 pF, has a delay time of 3.2 ns, occupies an area of 40,000 μm^2 and dissipates 175 mW of power. Heat flux was estimated at 14.2 W/cm^2, which leads us to believe that air cooling will be sufficient for this MCAP architecture implemented in MCM. [12]

6 Conclusions

The architecture to implement a class of high-performance interconnected ipelines attached processors, which can be modularly configured to match given sets of algorithms, has been presented. Performance analyses of typical application algorithms are given. The high utilization rate of processing components is achieved mainly by (1) minimizing the movement of intermediate results; (2) prefetching almost all operands using intelligent memory controller; and (3) reconfiguring (through programming) the interconnection of the processing components to

match the needs of a given algorithm. Based on the proposed architecture, we believe that an MCAP can be implemented in MCM using CMOS technology.

References

[1] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing,* McGraw-Hill, New York, 1985.

[2] J. A. Swanson, G. R. Cameron, and J. C. Haberland, "Adapting the ansys finite-element analysis program to an attached processor," *IEEE Computer*, vol. 16, no. 6, pp. 85–91, 1983.

[3] R. Hockney and C. Jesshope, *Parallel Computers 2*, Adam Hilger: Bristol, England, 1988.

[4] T. Hoshino, *PAX Computer: High-Speed Parallel Processing and Scientific Computing*, Addison-Wesley Publishing Company: Reading, Massachusetts, 1985.

[5] J. H. Tang and E. S. Davidson, "An evaluation of Cray I and Cray X-MP performance on vectorizable Livermore FORTRAN kernels," *Proc. 1984 Int'l Conf. on Supercomputing*, pp. 510–518, 1988.

[6] J. Dongarra, "Linear algebra libraries for high-performance computers: A personal perspective," *IEEE Parallel & Distributed Technology*, pp. 17–24, February 1993.

[7] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon, "Nas parallel benchmark results," *IEEE Parallel & Distributed Technology*, pp. 43–51, February 1993.

[8] D. DeFatta, J. Lucas, and W. Hodgkiss.

[9] D. Harold, "Superchips for supercomputing," *IEEE Spectrum*, vol. 29, no. 9, pp. 66–68, 1992.

[10] K. R. Nary and S. I. Long, "GaAs two-phas dynamic FET logic: a low-power logic familty for VLSI," *IEEE Journal of Solid-State Circuits.*, vol. 27, no. 10, pp. 1364–1371, October 1992.

[11] G. F. Watson, "Interconnections and packaging," *IEEE Spectrum*, vol. 29, no. 9, pp. 69–71, 1992.

[12] G. Gibson, , V. Singh, S. Singh, Y. cheng Liu, Y.-C. Chang, and S. Cabrera, "MCM implementation of modularly configurable attached processors," *Submitted for publication in 7th Int'l Conf. on Parallel and Distributed Computing Systems*, 1994.

An [illegible]

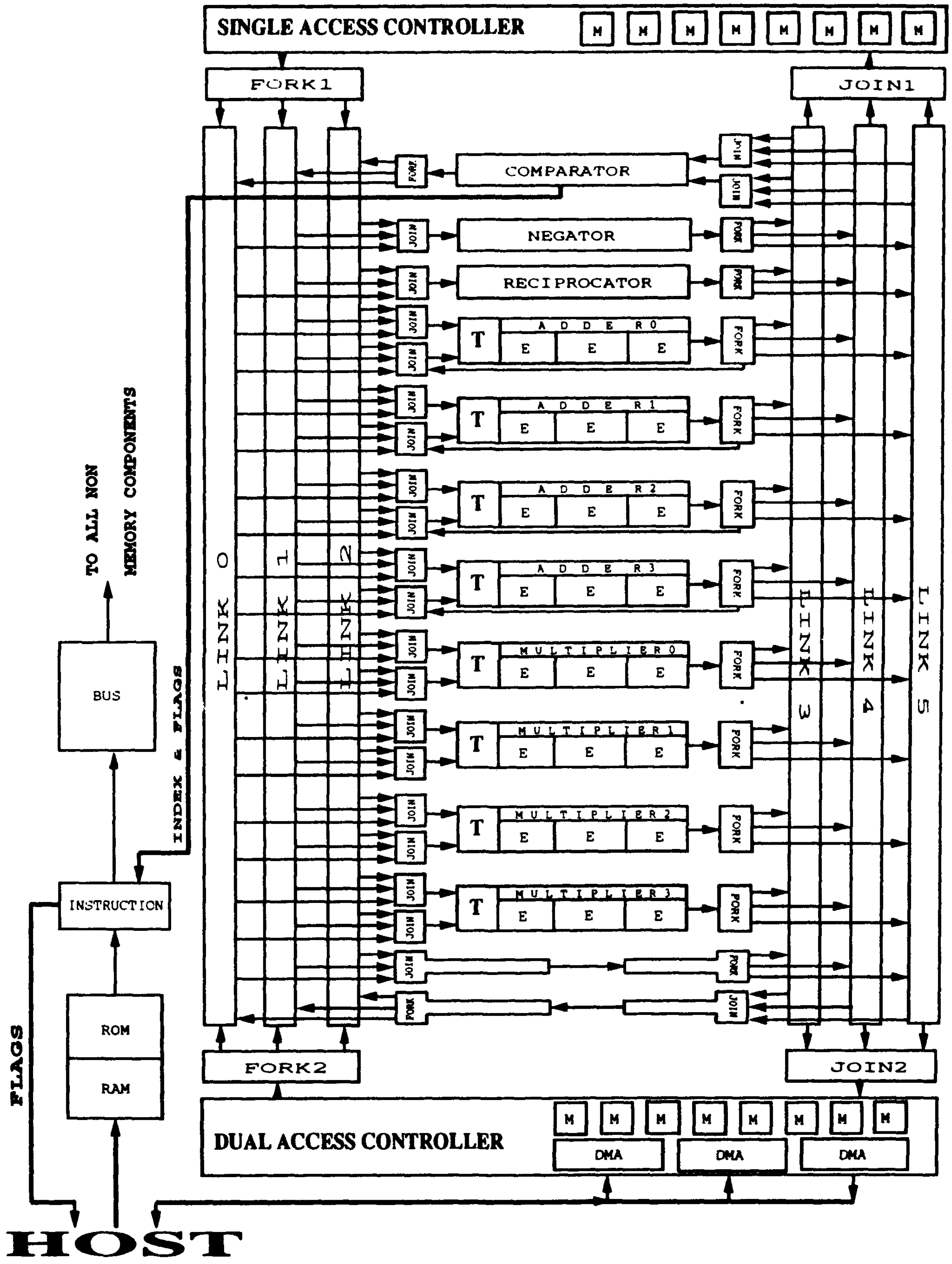

SINGLE ACCESS CONTROLLER
M
FORK1
JOIN1
LINK 0
LINK 1
LINK 2
LINK 3
LINK 4
LINK 5
COMPARATOR
NEGATOR
RECIPROCATOR
ADDER 0
ADDER 1
ADDER 2
ADDER 3
MULTIPLIER 0
MULTIPLIER 1
MULTIPLIER 2
MULTIPLIER 3
T
E
JOIN
FORK
FORK2
JOIN2
DUAL ACCESS CONTROLLER
DMA
TO ALL NON MEMORY COMPONENTS
BUS
INDEX & FLAGS
INSTRUCTION
ROM
RAM
FLAGS
HOST

FIGURE 1

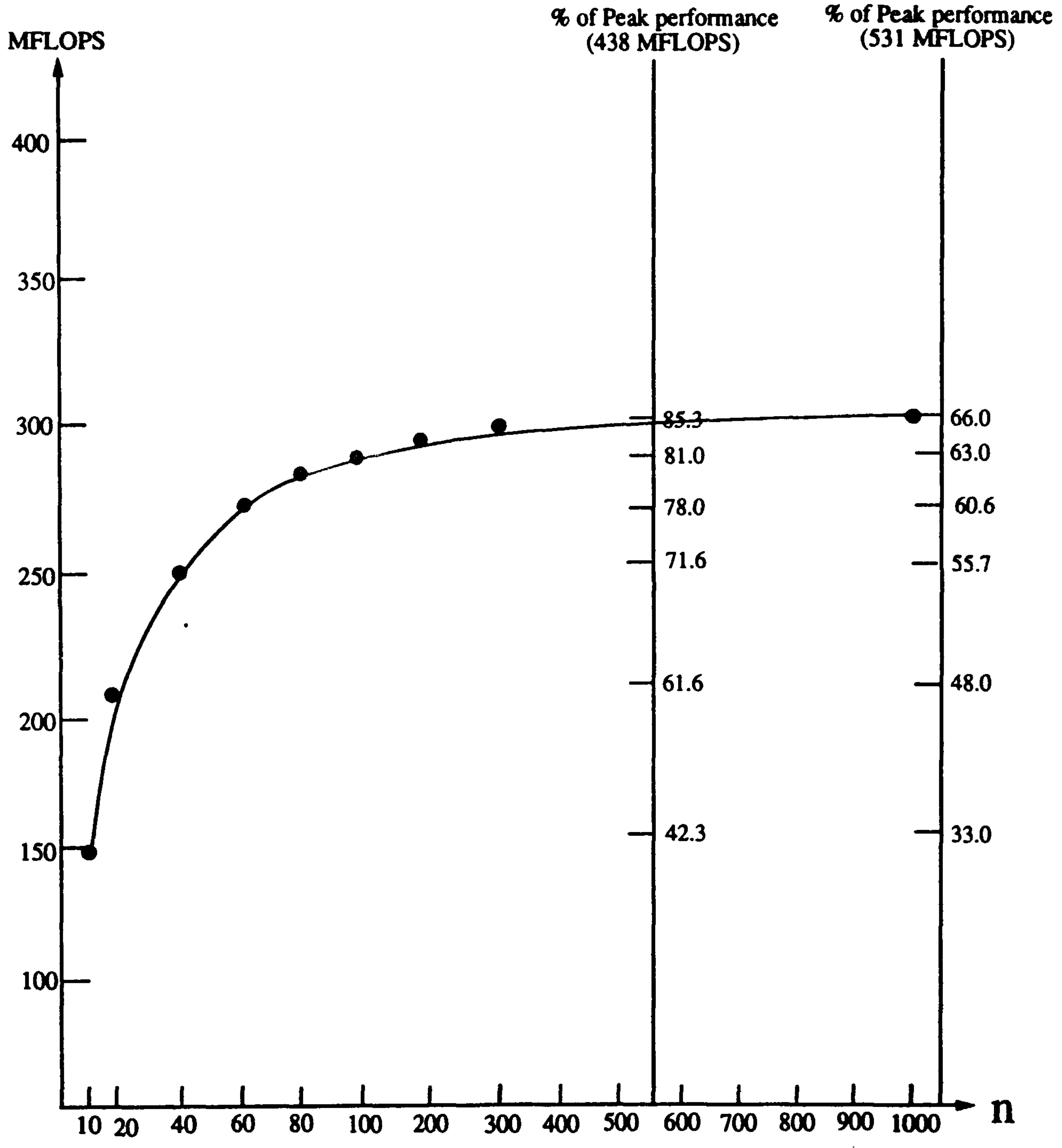
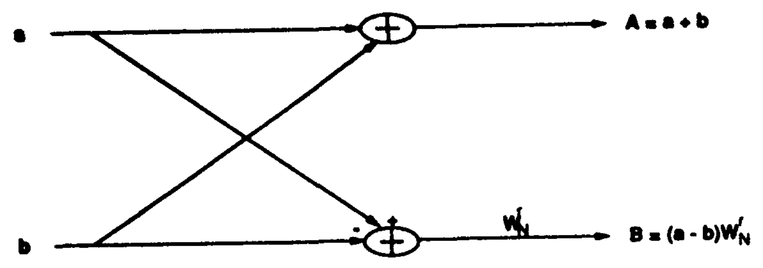
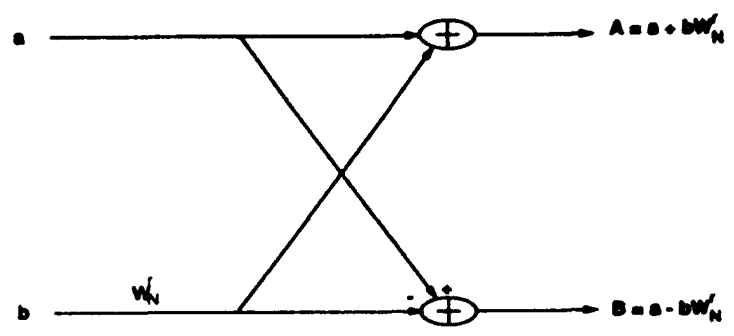


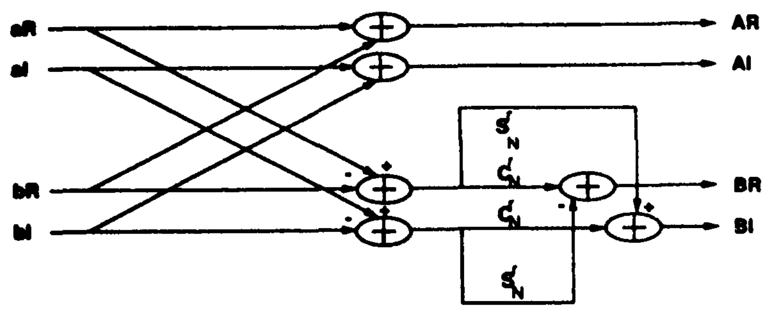
Fig. 2. Sustained performance vs. peak performance.



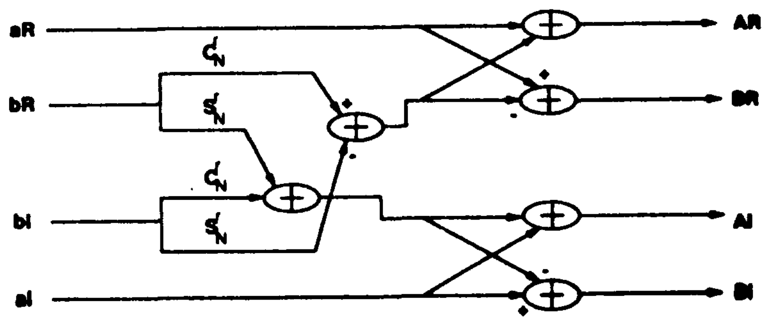
Basic Butterfly Computation in the Decimation-In-Frequency FFT Algorithm.



Basic Butterfly Computation in the Decimation-In-Time FFT Algorithm.

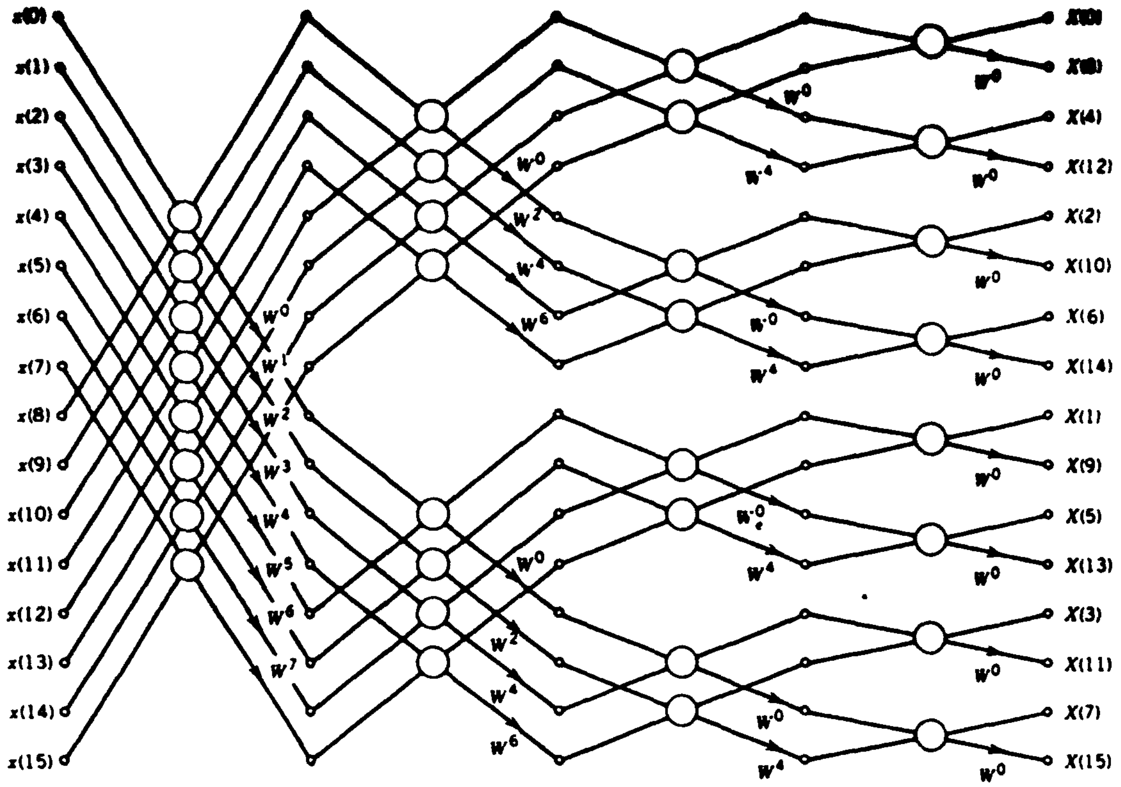


Basic Butterfly Computation in the DIF FFT Algorithm showing real and imaginary parts separately. Twiddle Factor $W'_N = C'_N jS'_N$

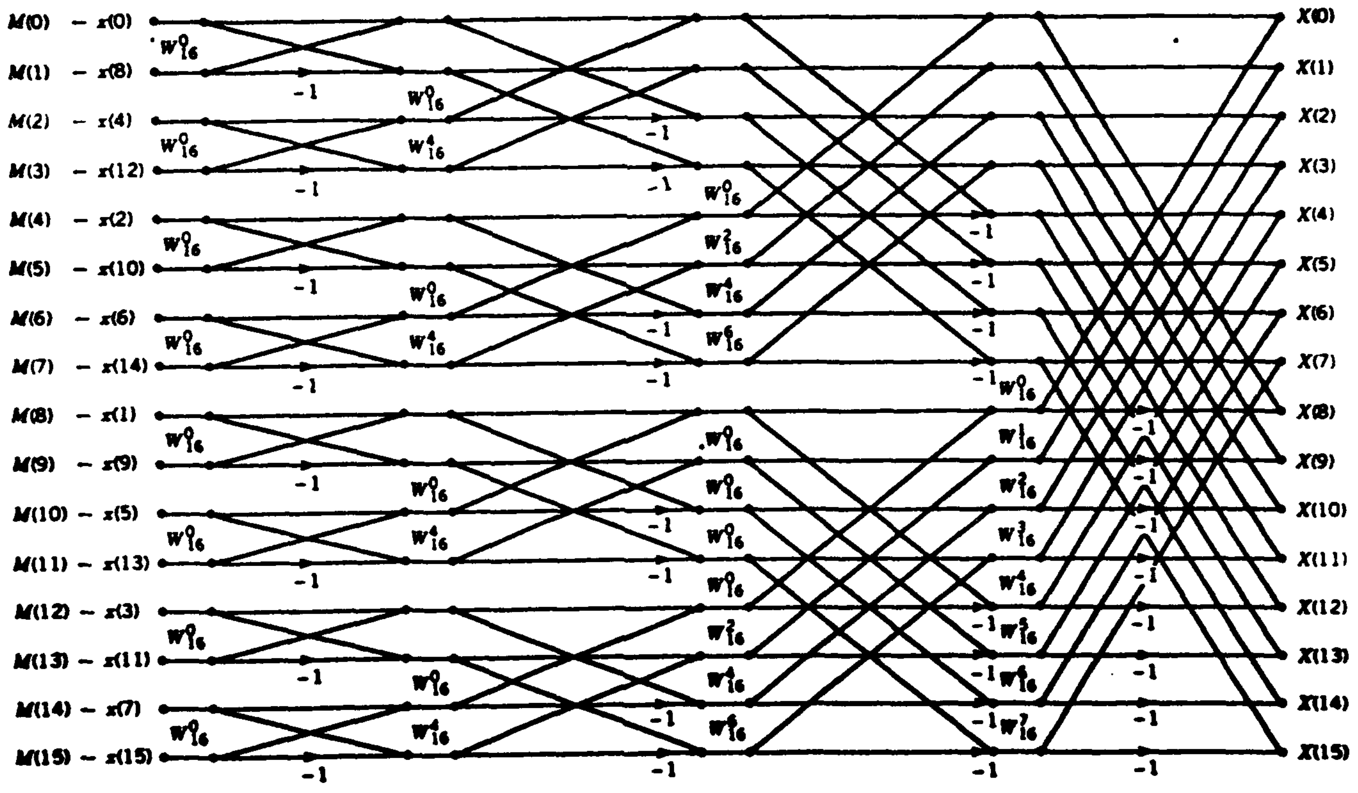


Basic Butterfly Computation in the DIT FFT Algorithm showing real and imaginary parts separately.

FIGURE 3



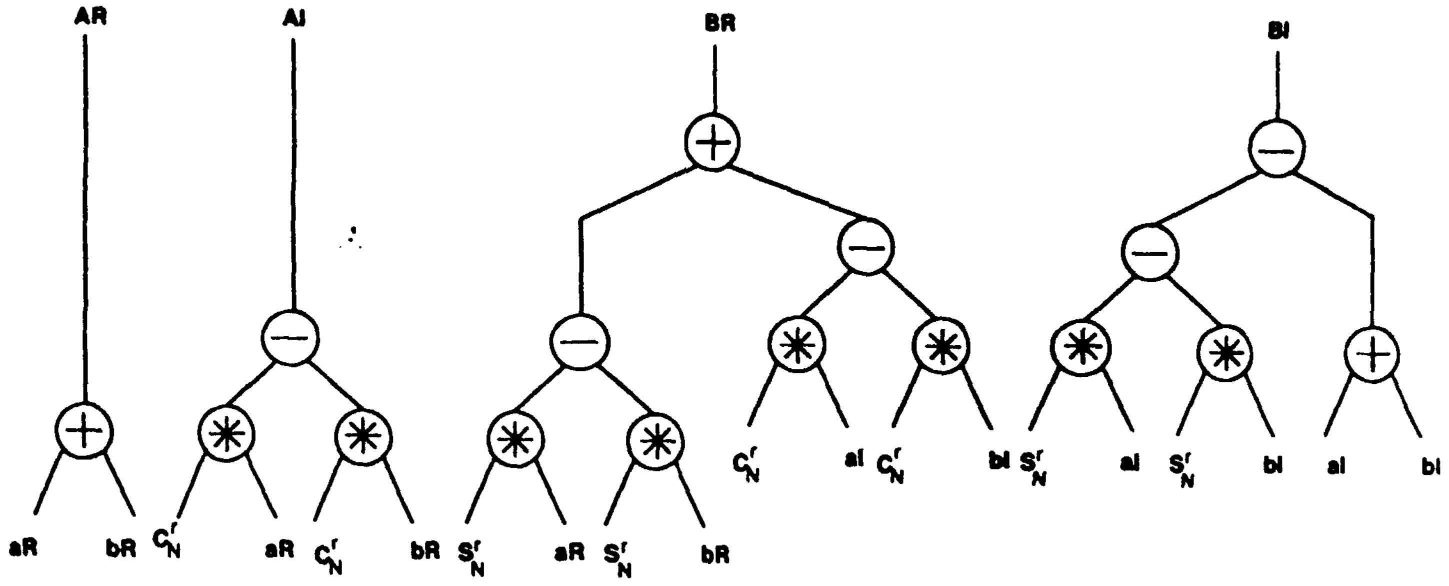
16-point DIF Radix-2 FFT flow diagram.



Flow graph of 16-point DIT FFT.

FIGURE 4

COMPUTATION EXPANSION TREE. FOR DIF FFT



COMPUTATION EXPANSION TREE. FOR DIT FFT

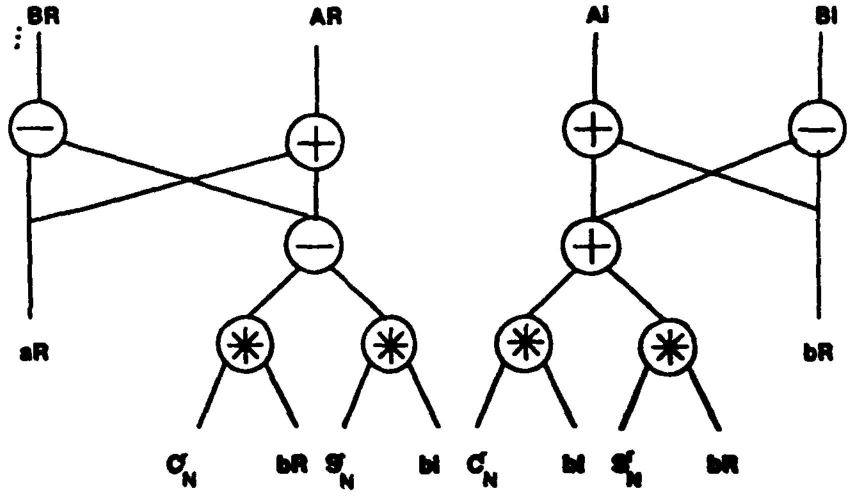
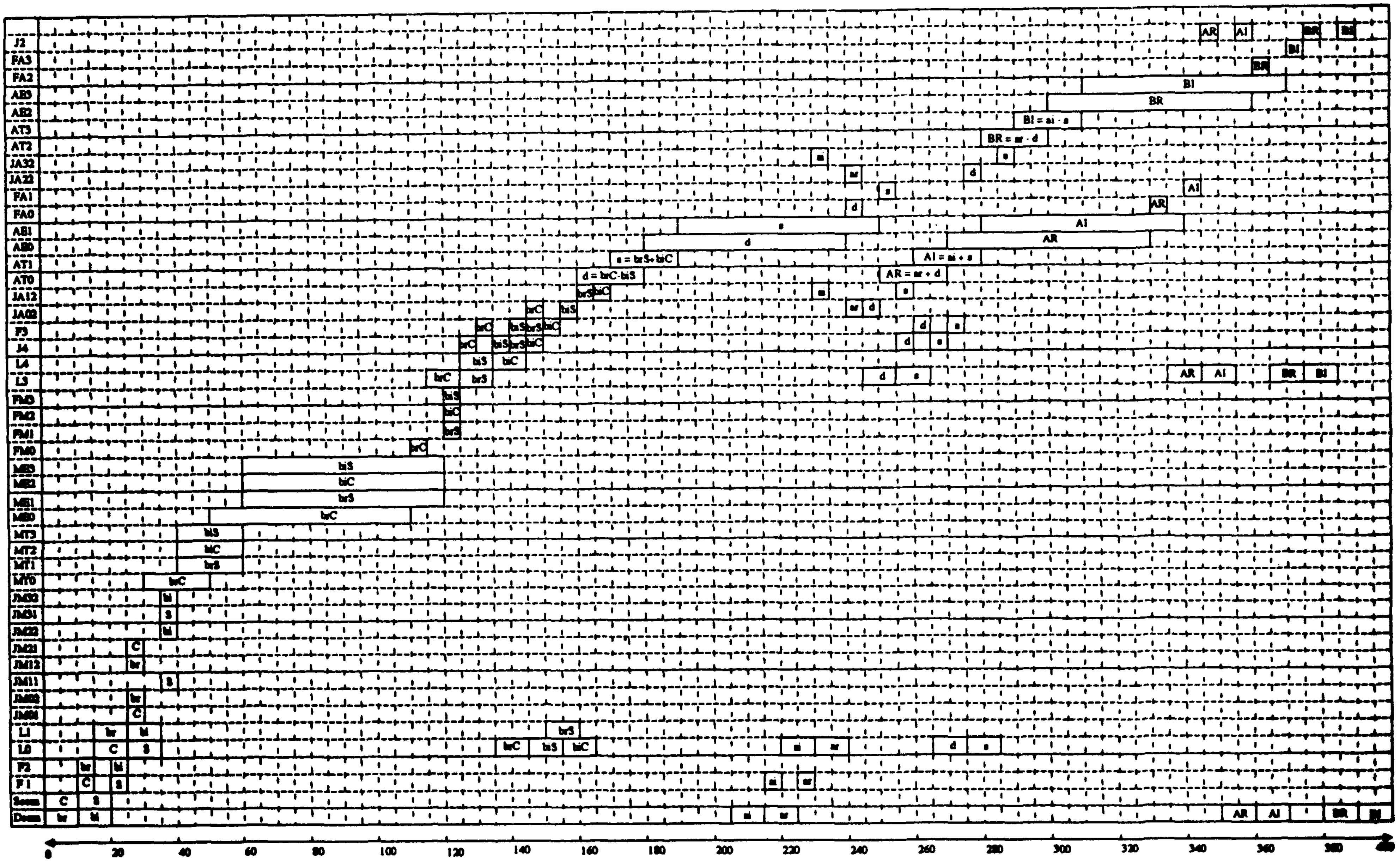
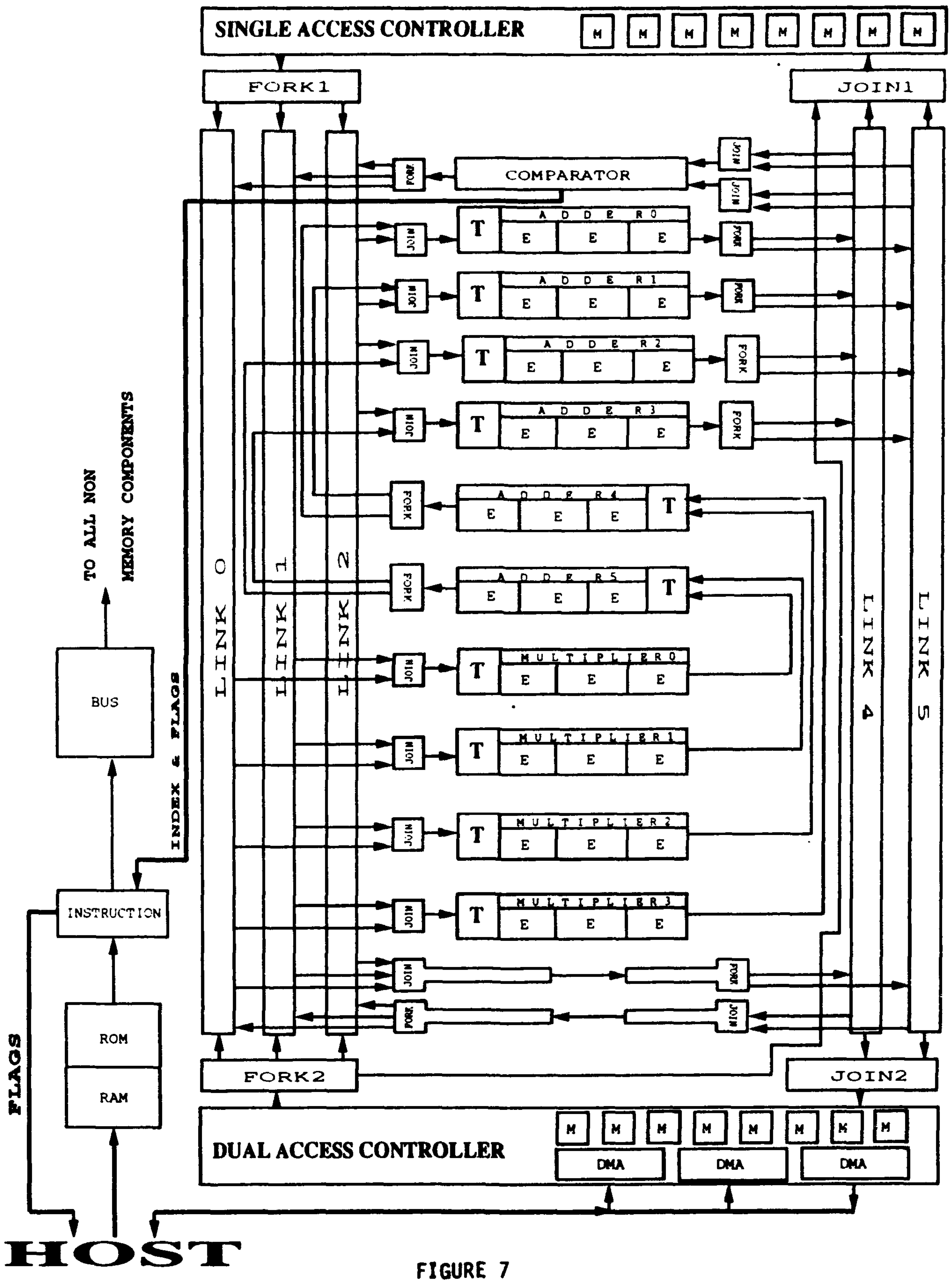


FIGURE 5



Data-flow and timing diagram of the DIT-FFT algorithm applied to the Arch#1

FIGURE 6



SINGLE ACCESS CONTROLLER
M
FORK1
JOIN1
LINK 0
LINK 1
LINK 2
LINK 4
LINK 5
COMPARATOR
FORK
JOIN
T
E
ADDER 0
ADDER 1
ADDER 2
ADDER 3
ADDER 4
ADDER 5
MULTIPLIER 0
MULTIPLIER 1
MULTIPLIER 2
MULTIPLIER 3
FORK2
JOIN2
DUAL ACCESS CONTROLLER
DMA
TO ALL NON MEMORY COMPONENTS
BUS
INDEX & FLAGS
INSTRUCTION
ROM
RAM
FLAGS
HOST

FIGURE 7

Timing Table for the Data Flow using the DIT-FFT Architecture

(one computation)

Description : $1 = C_N^f * bR$ $3 = S_N^f * bI$ $- = 1 - 3$

$2 = C_N^f * bI$ $4 = S_N^f * bR$ $+ = 2 + 4$

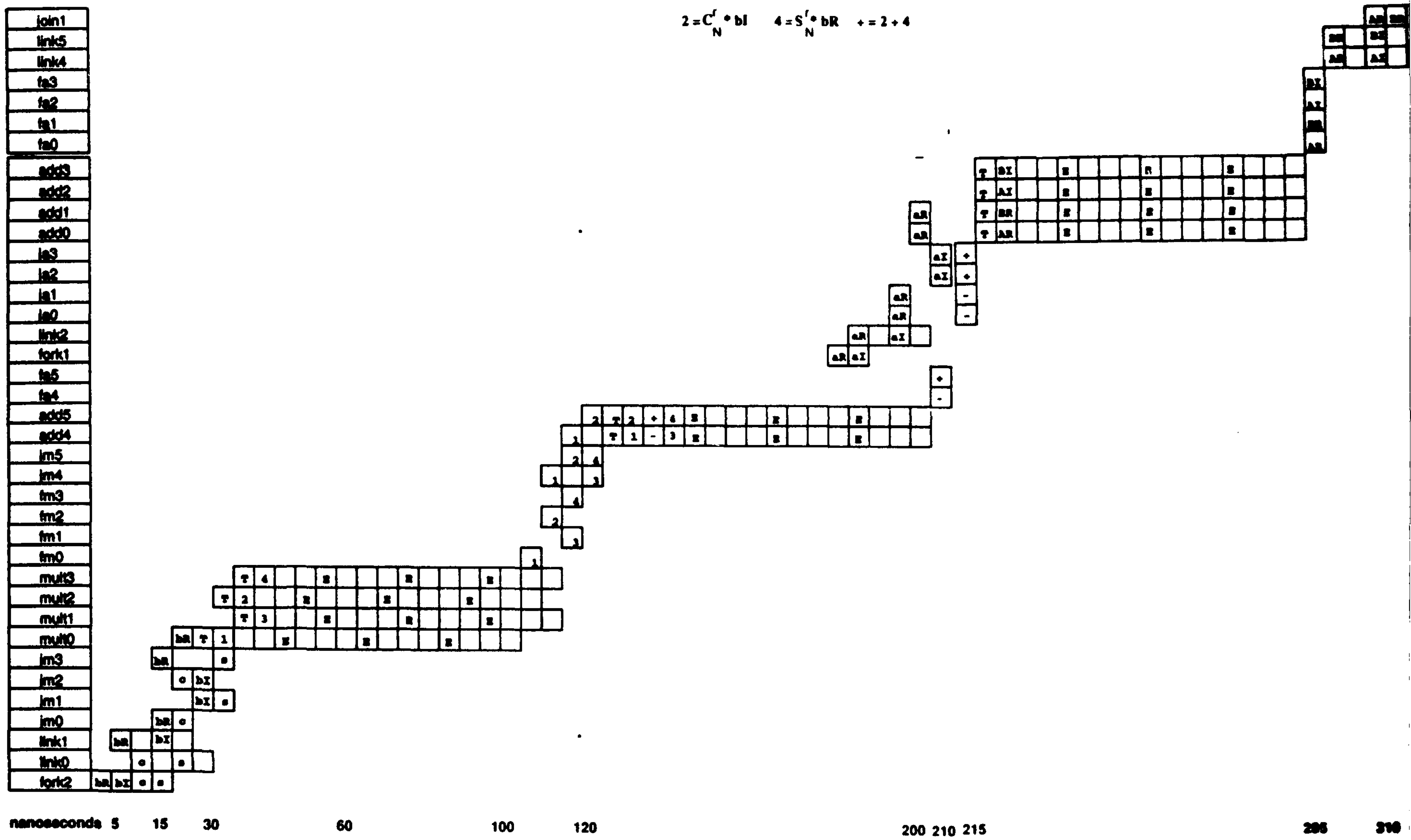
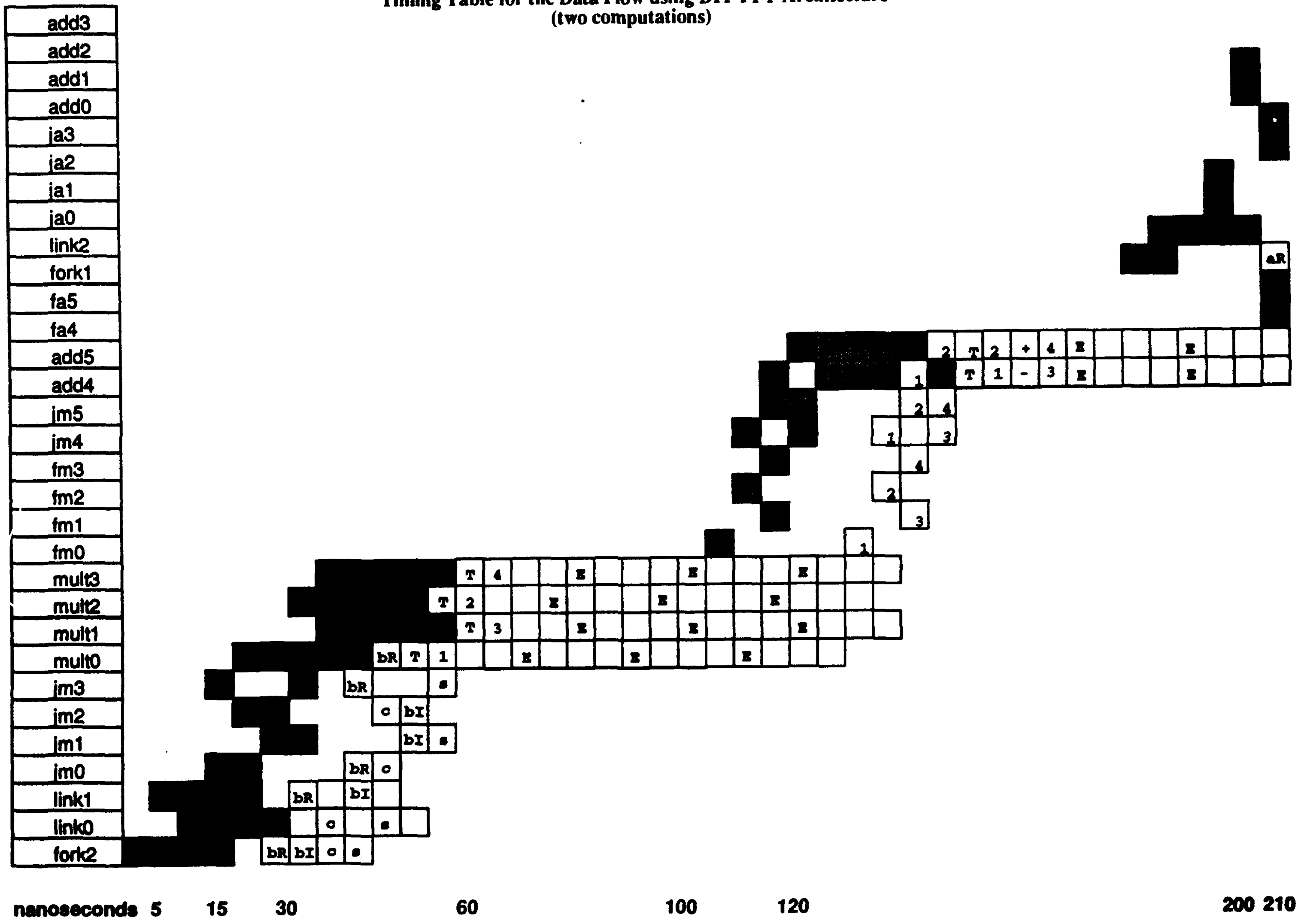


FIGURE 8

FIGURE 9



Timing Table for the Data Flow using DIT-FFT Architecture
(two computations)

FIGURE 9

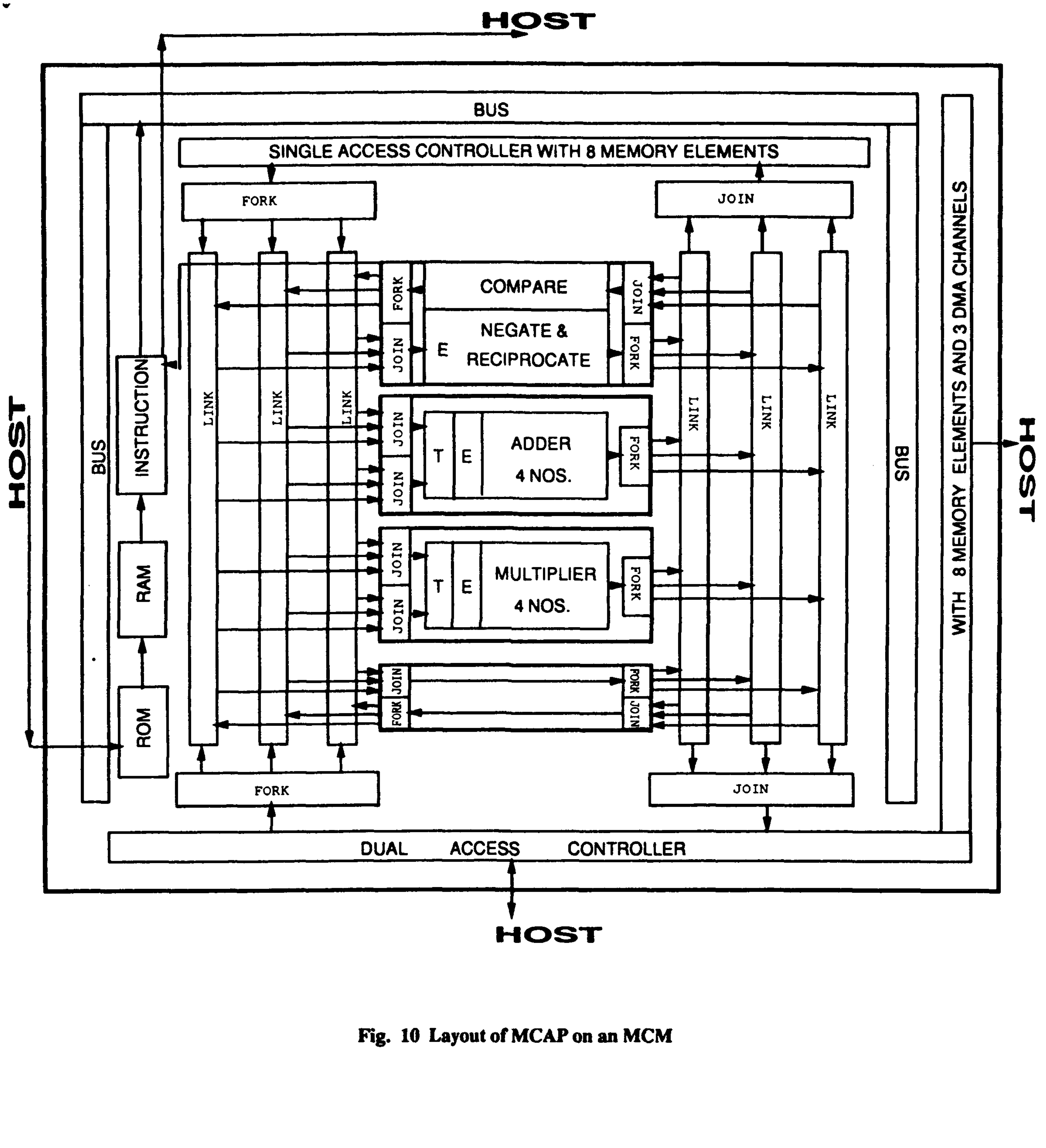


Fig. 10 Layout of MCAP on an MCM